Análisis y desarrollo de una infraestructura de containerización

Autor: Alberto Moragrega Láez
Director: César Hernández Baigorri, Technology 2 Client
Ponente: Juan José Costa Prats, Departamento de Arquitectura de Computadores
Especialidad: Ingeniería de computadores

Facultad de Informática de Barcelona
Universidad Politécnica de Cataluña
Technology 2 Client

3 de Julio de 2018

Resumen

Gracias a los avances en campos como el almacenamiento de datos, las redes o la capacidad de proceso, el campo de la informática ha experimentado un gran avance en las últimas décadas. Como ya predijo Moore [(1)] hace más de 50 años, el número de transistores en un microprocesador será del doble cada dos años y, a día de hoy, esto se sigue cumpliendo de forma bastante aproximada. Por el contrario, este desarrollo genera un mayor coste económico, lo cual lo hace accesible a muy pocos. Por ello, uno de los grandes retos actuales de la informática es optimizar el uso de recursos. En el campo de la virtualización, un ejemplo de ello es la containerización, que no sólo permite mayor eficiencia en costes debido al requerimiento de menos hardware y recursos humanos para despliegue y mantenimiento, sino que además consume menos energía al requerir menos complejidad.

Este proyecto propone un modelo de infraestructura diseñada para trabajar con contenedores. Para ello se ha partido de la herramienta Docker y se ha desarrollado un orquestador sobre dicha plataforma para poder realizar una gestión de los contenedores y mantener unos altos niveles de disponibilidad con un menor uso de recursos.

El objetivo del proyecto es determinar la viabilidad de migrar un gran volumen de servicios a este tipo de infraestructura. Para determinar dicha viabilidad, se realiza un estudio comparativo de rendimientos de las máquinas antes y después. La reducción de recursos usados por el Sistema Operativo de los servidores virtuales eliminados con esta metodología presenta por sí sola un ahorro en recursos disponibles. Adicionalmente, este modelo de infraestructura presenta una mejor respuesta a fallo ya que permite realizar la automatización de las migraciones de los contenedores a servidores virtuales distintos de forma rápida y eficaz. Por último, esta nueva tecnología permite una notable reducción adicional en licenciamiento de productos.

Resum

Gràcies als avenços en els camps com l'emmagatzemament de les dades, les xarxes o la capacitat de processament, el cap de la informàtica ha experimentat un gran avanç durant les últimes dècades. Com ja va predir Moore fa més de 50 anys, el nombre de transistors en un microprocessador serà del doble cada dos anys i, avui dia, això se segueix complint de forma bastant aproximada. Per altra banda, aquest desenvolupament genera un major cost econòmic, el qual el fa accessible només a uns pocs. Per això, un dels grans reptes actuals de la informàtica és optimitzar l'ús dels recursos. En el camp de la virtualització, un clar exemple és la containerització, que no tan sols permet una major eficiència en costs gràcies al requeriment de menys hardware i recursos humans per al desplegament i manteniment sinó que a més consumeix menys energia al requerir menor complexitat.

Aquest projecte proposa un model d'infraestructura dissenyat per treballar amb contenidors. Per fer-ho, s'ha partit de l'eina Docker i s'ha desenvolupat un orquestrador sobre aquesta plataforma per poder realitzar una gestió dels contenidors i mantenir uns alts nivells de disponibilitat amb un ús menor dels recursos.

L'objectiu del projecte és determinar la viabilitat de migrar un gran volum de serveis a aquest tipus d'infraestructura. Per determinar aquesta viabilitat, es realitza un estudi comparatiu de rendiments sobre les màquines abans i després. La reducció de recursos utilitzats pel Sistema Operatiu dels servidors virtuals eliminats amb aquesta metodologia presenta per si sola un estalvi en els recursos disponibles. Addicionalment, aquest model d'infraestructura presenta una millor tolerància a fallades, ja que permet realitzar l'automatització de les migracions dels contenidors a servidors virtuals diferents de forma ràpida i eficaç. Per últim, aquesta nova tecnologia permet una notable reducció addicional en el llicenciament de productes.

Abstract

Due to the progress in fields like data storage, networking or computational capability, computing has experimented a huge advancement in the last decades. As Moore said 50 years ago, the number of transistors in a microprocessor will double every two years and, nowadays, this is still valid in an quite accurate way. Because of this, this development increases the economic cost, making it available only for a few. Due to this, one of the greatest challenges of the computing nowadays relies in the optimization of the use of the hardware resources. In the field of virtualization, an example of this is the containerization, which not only allows a better efficiency in costs due to less hardware requirement and human resources for the deployment, but it also consumes less energy because of the requirement of less complexity.

This project proposes an infrastructure model designed to work with containers. For this, starting with the tool Docker we have developed an orchestrator running on this new infrastructure to provide management to the containers and maintain higher availability levels, all of this with less resources utilization.

The purpose of this project is to determine the viability to migrate a big amount of services to this kind of infrastructure. To determine the viability, we made a comparative study of the performance of the machines between the traditional infrastructure and the new infrastructure. The reduction in resources used by this new methodology shows a reduction in the available resources. Furthermore, this infrastructure model shows a better response against failure, since it allows the migration automatization of the containers to other virtual servers in a fast and effective way. Finally, this new technology shows a notable reduction in product licensing.

Reconocimientos

Este sin duda ha sido un trabajo en el que bastantes personas se han visto implicadas. Desde personas que han estado ahí desde el primer momento, hasta incorporaciones medio recorrido o esporádicas, todas han aportado su parte para que este proyecto pueda concluir con éxito. Quisiera agradecer la ayuda aportada a todas ellas.

Por parte de la universidad, agradecer a mi ponente, Juan José Costa, la colaboración con la idea y su interés desde el primer momento en realizar este proyecto con él.

Por parte de la empresa, agradecer a mi director, César Hernández, toda la ayuda aportada con el diseño y las entregas y, sobretodo, por haber estado detrás de mí evitando desastres. A Xavier Rodríguez y Lionel Magan, por su incalculable ayuda durante el desarrollo del proyecto y a Enric García, por brindarme desde el primer momento la oportunidad de realizar el proyecto en la empresa. Por supuesto, no puedo olvidarme del resto de mis compañeros de equipo, quienes se han preocupado por mí día tras día y se han mostrado dispuestos a ayudarme en cualquier momento.

Quisiera también agradecer a aquellos que me aportaron otro tipo de apoyo: el cariño y la motivación. Agradecer a mi grupo de amigos tanto de la universidad como del instituto, por todos esos buenos momentos juntos. A mis padres, Carmen y Oscar, y mi hermano, Alejandro, por haberme soportado y apoyado en todos esos momentos difíciles durante la carrera y el proyecto. No estaría donde estoy de no ser por ellos. Pero sobretodo, quisiera dedicar mi mejor agradecimiento a Elena, por esa felicidad y esa paz que me transmite cada momento junto a ella, que hace que todo sea increíblemente más sencillo.

Índice de contenidos

Índice de figuras

1. Introducción

1.1. INTRODUCCIÓN AL PROYECTO

Desde los inicios de la informática, muchas empresas empezaron a invertir en tecnología para sus negocios. En 1960, IBM empezó a introducir el concepto de virtualización [(2)] en su hardware, pero éste no tuvo mucho éxito, pues aún existía cierta desconfianza. No fue hasta la década de los 90 cuando se empezó a observar que debido a la evolución del hardware del que se disponía empezaba a haber recursos desaprovechados. Fue entonces cuando la virtualización comenzó a ganar fuerza como una manera de aprovechar dichos recursos y de reducir costes, espacio, etc. A partir de ese momento, la virtualización comenzó a evolucionar y hoy en día es la tecnología más utilizada en los datacenters de todo el mundo. El concepto de containerización intenta ir un paso más allá.

El campo de la containerización se orienta a un menor uso de recursos y un menor tiempo de despliegue. Entornos que requieren de un elevado número de servidores en una misma máquina ven una clara reducción de recursos en la misma gracias a esta tecnología. La containerización proporciona completa independencia a las aplicaciones del sistema operativo que ejecuta la máquina. Estos contenedores ejecutan la imagen del sistema operativo que las aplicaciones requieren incluyendo todos los paquetes y dependencias necesarias para su uso. Dichos sistemas operativos son lo más ligeros posibles, ya que lo que se pretende es un menor uso de los recursos con tal de que estos puedan ser destinados a otras funcionalidades. Es decir, esta tecnología permite que con el mismo hardware y sin invertir grandes cantidades de dinero se puedan aprovechar aún más los recursos de las máquinas.

Tanto en el contexto empresarial como en el ámbito de la investigación, este proyecto analiza el rendimiento de la virtualización de las aplicaciones y de la infraestructura diseñada. La containerización es una tecnología en auge y estudiar su viabilidad es sin duda de gran importancia para el futuro. La containerización es un método cada vez más frecuente en empresas [(3)], como por ejemplo Paypal o Visa.

1.2. ESTADO DEL ARTE

1.2.1. Historia de la containerización

Las primeras aproximaciones [4] de lo que hoy se conoce como containerización empezaron en 1979 con Unix V7 [5]. Esta versión del sistema operativo Unix fue la primera en introducir la llamada a sistema chroot, la cual permitía cambiar el directorio root de un proceso y del de sus hijos a una nueva localización en el sistema de ficheros. Este avance fue considerado el inicio de lo que se conoce como "process isolation", es decir, el aislamiento de procesos, el cual segrega el acceso a los ficheros de cada proceso. En 1982, chroot fue añadido al BSD

En el año 2000, FreeBSD [6] creó lo que nombró como "FreeBSD Jails" (Celdas en castellano). Esta herramienta permitía a los administradores particionar el sistema operativo en diversos sistemas más pequeños, llamados celdas, los cuales eran completamente independientes, siendo incluso capaces de asignarles una IP y una configuración distinta a cada uno. Esta funcionalidad empieza a parecerse más a lo que hoy se conoce como contenedor.

En 2001 nace una distribución experimental de Linux que incluye el mismo concepto que las Jails de FreeBSD llamada "Linux Vserver" [7]. Linux VServer implementa más o menos las mismas funcionalidades en cuanto a particionado de recursos que FreeBSD. No fue hasta 2008 que se empezaron a lanzar los primeros parches estables de esta distribución.

En 2004 aparecen los "Solaris Containers" [8], los cuales combinan el control de recursos con la separación por zonas (Concepto introducido por ellos, el cual se refiere a un entorno aislado y seguro para ejecutar aplicaciones). Esto permitió la creación de funcionalidades como los snapshots y el clonado de ZFS[1]

En 2005 se lanzó "Open VZ" [9]. Open Virtuozzo es una herramienta de virtualización a nivel de Sistema Operativo para Linux, el cual utiliza el propio kernel de Linux para la virtualización, aislamiento y gestión de los recursos.

En 2006, Google lanza lo que se conoce como "Process Containers" [10]. Esta herramienta estaba diseñada para limitar y aislar el uso de recursos (CPU, Memoria, I/O a disco, network, etc) de un conjunto de procesos. En 2007 se renombró como "Control Groups" (cgroups) [10] y pasó a formar parte del kernel de Linux.

[1] Zettabyte File System, es un sistema de archivos y volúmenes desarrollado por Sun Microsystems para su sistema operativo Solaris.

Por fin, en el año 2008 nace la implementación más cercana por el momento al concepto de containerización que conocemos actualmente: “Linux Containers” (LXC) [11]. Esta herramienta se implementó utilizando cgroups y el aislamiento de espacio de nombres propio de Linux. LXC permite que un servidor físico ejecute múltiples instancias de sistemas operativos aislados. Es decir, introduce la idea de máquinas virtuales.

En 2011 nació “Warden”, de CloudFoundry [12]. Utilizando las primeras implementaciones de Linux Containers, las cuales luego cambió por las suyas propias, Warden permite aislar entornos en cualquier sistema operativo, ya que se ejecuta como un Daemon en la máquina y provee una API para su gestión.

En 2013 apareció “Let Me Contain That For You (LMCTFY) [13] como una versión de código abierto de los contenedores de Google “Process Container”. Esta herramienta introducía la novedad de que los contenedores podian ser “container aware”, es decir, se les daba la posibilidad de crear y gestionar sus propios subcontenedores. Este proyecto cesó su desarrollo cuando empezó a colaborar con Google en lo que hoy se conoce como “Open Container Foundation”

Finalmente, en 2013 nace “Docker” [14]. Gracias a esta herramienta, los contenedores ganaron muchísima popularidad. Como Warden, Docker también empezó utilizando las primeras implementaciones de Linux Containers, que rápidamente fueron sustituidas por su propia librería, llamada libcontainer. Lo hizo a Docker destacar y ser diferente del resto de herramientas fue que ofreció un ecosistema completo de gestión de contenedores.

Desde entonces hasta ahora, las herramientas de gestión de contenedores han ido evolucionando, sobretodo en términos de seguridad y nuevas funcionalidades.

1.2.2. Historia y recepción de Docker

Salomón Hykes comenzó a desarrollar Docker [15] como un proyecto interno en la compañía dotCloud, la cual en aquel momento se trata de una compañía enfocada a la plataforma como servicio, con contribuciones de otros ingenieros de la misma empresa. El 13 de marzo de 2013 se lanzó Docker. Un año más tarde, Docker reemplazó Linux Containers como entorno de ejecución por el suyo propio, el cual nombró “libcontainer” y estaba escrito en lenguaje Go. Un año más tarde, el

proyecto fue tan popular, que tenía más de 20.700 estrellas en Github, más de 4.700 forks y casi 900 colaboradores. Esto claramente favoreció su popularidad.

1.2.3. La containerización en la actualidad

Actualmente, la virtualización es un tema del cual existen infinidad de artículos académicos con nuevas propuestas, estudios, análisis, usos, etc. sobre virtualización.

Centrándonos en el objetivo de este proyecto, gracias a la amplia comunidad de Docker, podemos encontrar contenedores de muchísimas aplicaciones, ya sean distribuciones oficiales de las mismas como apache, mongodb, ubuntu, etc. o muchísimos otros contenedores creados por los usuarios para múltiples propósitos, como videojuegos, gestión de servicios, bots, etc.

Esto hace que una infraestructura como la que se pretende realizar para la empresa sin duda ya exista, y de distribuidores conocidos, como Google y su *Kubernetes* [16] o no tan conocidos, como el software de monitorización de contenedores *Docker Rancher* [17], pero que, a su vez, hace que se facilite la tarea de desarrollar una nueva, ya que pueden ser tomadas como referencia. Estos ejemplos se detallan a continuación:

- Kubernetes nació en 2014. Es una aplicación de código abierto escrita en Go desarrollada por Google que permite la automatización del despliegue y la gestión de los diversos contenedores.
- Docker Rancher es un software de código libre creado por Rancher Labs en 2014 utilizado para la gestión y monitorización de los contenedores Docker.

1.3. CONTEXTO

Este proyecto consta de unos actores implicados de forma directa, algunos otros actores que actúan de forma puntual y, por supuesto, unos beneficiarios finales. Todos ellos se detallan a continuación.

1.3.1. Actores directos

El desarrollador

En este caso yo, Alberto Moragrega. Es la persona que ha llevado a cabo todo el desarrollo del proyecto. Esto incluye tanto programación de los diferentes componentes, como de la gestión del proyecto, el escrito de la memoria, preparar presentaciones, adecuarse a las entregas pactadas, etc. Para todo esto, cuenta, con la ayuda de los siguientes actores.

El director

El tutor por parte de la empresa Technology 2 Client, en este caso, César Hernández. Decide qué pasos se han de tomar y hacia donde se ha de encaminar el desarrollo para que se adecue a las necesidades de la empresa o suponga un paso para la innovación.

DevOps

El personal de soporte de la empresa Technology 2 Client. Ayuda con el desarrollo del proyecto desplegando el entorno y proveyendo ayuda, consejos y soporte para el mismo.

El ponente

El tutor por parte de la Facultad de Informática de Barcelona, en este caso, Juan José Costa, coordinador y profesor de la especialidad de Ingeniería de Computadores. Su rol consiste en el soporte en cuanto al desarrollo del proyecto, resolución de dudas, consejos, y hacer que el trabajo se ajuste a la normativa académica de la facultad.

El profesor de GEP

El tutor por parte del módulo de GEP, en este caso, Fernando Barrabes. Durante el módulo previo al TFG, GEP, se empieza a preparar el escrito de la memoria del proyecto. El rol de este actor es, pues, aconsejar sobre buenas prácticas de redactado de documentos, así como de presentación de proyectos.

1.3.2. Actores indirectos

Empleados de la empresa

Otros empleados que puedan dar soporte al proyecto. En la empresa existen diversas personas con conocimientos que pueden aplicarse al proyecto en mayor o menor medida. La ayuda de estos actores es vital para el desarrollo de todo el entorno del proyecto.

Beneficiarios

Los beneficiarios en este caso sería la empresa, ya que la supuesta puesta en producción de este proyecto, en caso de que el estudio de viabilidad la considerase válido, supondría un ahorro en cuanto a costes destinados a servidores. Además, como ya se ha detallado antes, la containerización de las aplicaciones ofrece una mejora no solo en rendimiento si no en fiabilidad, lo cual se traduce en un aumento de la tolerancia a fallos que implica también un beneficio para la empresa.

2. Objetivos y alcance

2.1. FORMULACIÓN DEL PROBLEMA

El crecimiento de demanda de recursos de las empresas hace que sea requerido un mayor o mejor hardware, lo cual deriva en un aumento de los costes, tanto los inducidos por la compra del nuevo hardware como los que se extraen del mantenimiento de los mismos, como por ejemplo el coste eléctrico de mantenerlos funcionando.

En su momento, esto fue resuelto con la virtualización de los sistemas operativos en máquinas virtuales, pero hoy en día esto puede llegar a no ser suficiente. Existen empresas que disponen de diversas máquinas virtuales en un mismo servidor físico, lo cual hace que el uso que se les da a dichas máquinas físicas sea considerablemente grande. Aquí es donde entra la containerización de aplicaciones.

En el contexto de la empresa en la que trabajo, la mayoría sistemas de la empresa corren sobre VMware. Esto supone que para web apps con requisitos especiales, como por ejemplo distintas versiones de Java o PHP, es necesaria una máquina virtual dedicada. Esto hace que se requiera un coste económico mayor derivado de los recursos necesarios para llevar a cabo esa infraestructura y del consumo energético de los mismos.

2.2. OBJETIVOS

El objetivo principal de este proyecto es proponer una solución al problema indicado mediante la containerización de las aplicaciones utilizado la herramienta Docker. Para ello, se han marcado los siguientes subobjetivos:

2.2.1. Containerización de aplicaciones

Mediante la containerización de las aplicaciones se consigue, entre otros beneficios, una independencia de la aplicación con el Sistema Operativo. Esto hace que en una misma máquina virtual en un mismo servidor físico puedan correr todas las aplicaciones. Con esto, simplemente con una redirección según el puerto en cuestión, sería posible acceder a cada aplicación web. Esta mejora, además de un menor tiempo de despliegue, implicaría una drástica reducción del número de máquinas virtuales lo que deriva en un menor consumo por parte de los servidores físicos e incluso la probabilidad de mantener algunos apagados o aprovecharlos para llevar a cabo otras funciones.

2.2.2. Infraestructura

Para poder obtener un mayor rendimiento de esta mejora, se desarrolla una infraestructura consciente de esta nueva implementación. Ésta consiste en dos nodos en los cuales se ejecutan los contenedores, ambos conectados a otras máquinas en las cuales se almacenan los datos. Los dos nodos están conectados a una cuarta máquina en la cual se ejecuta un orquestador conjuntamente con su interfaz gráfica de control, con tal de que este nodo tenga acceso a los otros dos para poder realizar la gestión de los contenedores. El hecho de que los datos se encuentren fuera de los nodos Docker facilita las migraciones de los contenedores y permite que dicha estructura sea escalable.

2.2.3. Orquestador

Para tener una gestión y monitorización de los contenedores en esta nueva infraestructura, se desarrolla un orquestador. Este programa implementa las siguientes funcionalidades:

- Listado de contenedores activos por nodo.
- Arranque y parada de contenedores bajo demanda.
- Migración bajo demanda y automática en caso de posible caída del nodo o saturación del mismo.
- Monitorización, gráficos e información detallada de cada contenedor.
- Monitorización y gráficos de los nodos.

2.2.4. Interfaz web

Con tal que el orquestador sea más amigable para el usuario, se desarrollada una interfaz web. En esta interfaz se muestra la lista de contenedores por nodo, así como las estadísticas de los mismos de forma rápida y visual. Además, permite su gestión mediante botones de arranque, parada y migración.

2.3.5. Análisis de rendimiento

Por último, para probar la viabilidad de esta nueva propuesta, se efectúa un análisis del rendimiento de las máquinas antes y después de esta implementación. Estos datos se muestran de forma gráfica, facilitando su lectura. A continuación, se realiza el estudio de los mismos, con las conclusiones pertinentes.

2.3. ALCANCE

El proyecto consiste en la implementación de los cinco objetivos citados en el punto anterior. Una vez realizados, se da por finalizado cuando la infraestructura funciona como se espera. Esto es, que permite, como mínimo, tanto la automatización de la containerización de las aplicaciones, como la migración de las mismas. Esto conlleva que se han conseguido containerizar unas cuantas aplicaciones previamente. Para que se pueda dar por finalizado con éxito el proyecto, dicha infraestructura debe reportar algún tipo de beneficio en rendimiento o en coste económico. En caso de múltiples intentos y de no haber reportado ningún tipo de beneficio, el proyecto se dará por finalizado explicitando donde han surgido los distintos problemas para que los resultados del rendimiento no sean satisfactorios.

3. Metodología y rigor

3.1. HERRAMIENTAS

Para el desarrollo de este proyecto serán necesarias diversas herramientas. Tanto aplicaciones como los lenguajes de programación utilizados se definirán a continuación:

Docker (14)

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo en Linux. Esto evita la sobrecarga de realizar máquinas virtuales únicamente dedicadas a una aplicación.

Es sin duda una de las herramientas principales para el desarrollo del proyecto, puesto que es esencial para la containerización de aplicaciones. Su gran comunidad de usuarios hace que sea un entorno de desarrollo muy amplio y con múltiples posibilidades.

VMware (18)

VMware es un software de virtualización que ofrece la posibilidad de virtualizar diversos sistemas operativos para todo tipo de máquinas, tanto ordenadores personales como servidores de empresas.

Gracias a VMWare dispondremos de un entorno de desarrollo aislado. Además del desarrollo, las pruebas de uso de la infraestructura desarrollada, así como el análisis del rendimiento que esto supone, se realizan en dichos servidores virtuales.

CentOS 7 (19)

Siglas de Community ENTerprise Operating System, es una distribución de linux orientado a empresas. Es un sistema operativo de código abierto, basado en la distribución Red Hat Enterprise de Linux. Éste es el sistema operativo utilizado tanto para el desarrollo como para la puesta en producción de la infraestructura diseñada.

Putty [20]

Putty es un software libre para establecer una comunicación ssh con un servidor, utilizando windows. Originalmente, solo se encontraba disponible para windows, pero ahora, gracias a la aportación de los usuarios, está siendo desarrollado para unix y mac.

Google Drive [21]

El servicio de almacenamiento en cloud de google. Utilizado para copias de seguridad y desarrollo de documentos de entregas, así como para definir elementos a desarrollar y fechas para hitos del proyecto.

WinSCP [22]

Software gratuito para realizar envíos por FTP entre máquinas. Es especialmente útil con sistemas basados en linux en los cuales solo se dispone de un intérprete de comandos, puesto que facilita la gestión de los ficheros de forma gráfica.

Sonic Wall [23]

Software utilizado para conectarse a los recursos de la empresa desde una red externa vía VPN. Se utiliza con el usuario y los datos proporcionados por la propia empresa.

Python (2.7) [24]

Lenguaje de programación en el cual está escrito el código del orquestador, ya que la API oficial de Docker se encuentra disponible en este lenguaje.

MySQL [25]

Aplicación de gestión de Base de Datos utilizada para el envío y respuesta de datos del orquestador y la aplicación web.

HTML [26]**, PHP** [27]**, CSS** [28]**, Javascript** [29]

Lenguajes utilizados para la creación de la página web que servirá de interfaz gráfica del orquestador.

Trello [30]

Software para la gestión de tareas en proyectos con interfaz web y cliente para Android e iOS. Consiste en un tablón en el cual se añaden tarjetas con las tareas a realizar. Esta metodología está basada en el sistema Kanban o Just in Time.

Vcenter (31)

Software para la creación y gestión de las máquinas virtuales. Permite, entre otras muchas funcionalidades, una monitorización de los datos de los recursos de las máquinas. Esta funcionalidad es la que se ha utilizado para el estudio estadístico.

3.2. MÉTODO DE TRABAJO

El método de trabajo seguido corresponde con la denominada *metodología ágil*. El proyecto se empieza basándose en una idea principal, la cual va desarrollándose de forma iterativa e incremental con cada reunión con el director, aportando al proyecto nuevas necesidades o ideas. En ese momento, se tratan de implementar esas novedades hasta la próxima reunión, en la cual se valora si han sido satisfactorias o por el contrario se han de proponer alternativas.

3.3. MÉTODO DE EVALUACIÓN

El método de evaluación consiste en el estudio de una mejoría por parte del rendimiento de la máquina. Es decir, aparte de que los resultados del funcionamiento de los contenedores y del orquestador desarrollado sean los correctos, esta nueva infraestructura debe reportar beneficios para que sea viable en su posible puesta en producción. Para ello, se toman medidas de rendimiento de la máquina antes y después de la implementación de dicha nueva infraestructura.

4. Fases del desarrollo

4.1. RESUMEN DE LAS FASES DEL DESARROLLO

A continuación, se muestra una tabla resumen de las fases en las cuales se ha dividido el proyecto. Se detalla también, además de una descripción, el tiempo invertido en cada una con tal de cumplir con las fechas. Los recursos empleados en las fases son siempre los mismos, es decir, servidores con máquinas virtuales y el software necesario.

FASE	DESCRIPCIÓN	DURACIÓN
1. Preparación	En esta fase se hacen las preparaciones iniciales para el desarrollo del proyecto, tales como estudios, planificaciones y preparaciones de los entornos para poder pasar al desarrollo.	4 Semanas
2. Ejecución	En esta fase se desarrolla el proyecto. Ésta es una fase iterativa conjuntamente con la siguiente fase, evaluación. En ella se desarrollan tanto la containerización de las aplicaciones como el orquestador para gestionarlas.	10 Semanas
3. Evaluación	En esta fase se evalúan los resultados del desarrollo. Tanto esta fase como la anterior son iterativas, por tanto, si el resultado de la evaluación no es satisfactorio, se debe volver a la anterior fase para la corrección de los puntos que no han resultado como se esperaba.	10 Semanas
4. Go-Live	Cuando los resultados obtenidos son los esperados se pasa a esta fase, en la que el proyecto estaría listo para su posible puesta en producción.	1 Semana
5. Memoria	Durante el desarrollo del proyecto se debe ir redactando una memoria con todos los pasos, descubrimientos, cambios, funcionalidades, resultados, etc. del proyecto, con tal de que pueda ser reproducido en un futuro.	18 Semanas

Figura 1: Resumen de las fases del desarrollo

4.2. DETALLES DE LAS FASES DEL DESARROLLO

Independientemente de la metodología utilizada para el desarrollo del proyecto, es recomendable seguir unas fases. Para este proyecto, se han definido las detalladas a continuación:

1. Preparación

1.1 Planificación

En esta fase se realizan las planificaciones de las fases del desarrollo con tal de adaptarse a las fechas esperadas de las entregas. Esto además permite una mejor visualización del trabajo, el tiempo requerido y el tiempo restante.

1.2 Estudio de la aplicabilidad

Primeramente, se debe realizar un estudio para comprobar la viabilidad de la idea. En este estudio se valorará tanto la rentabilidad de la idea para la empresa, como el tiempo invertido por parte del desarrollador respecto al beneficio en rendimiento que se pueda sacar.

1.3 Estudio de la tecnología

Es necesario realizar un estudio de la tecnología disponible, tanto a nivel de hardware como de software, es decir, hasta donde se puede llegar con el hardware que se dispone y el software que se va a utilizar.

1.4 Definición de las métricas

Uno de los objetivos finales del proyecto es el estudio de la mejora o no en rendimiento de la propuesta. Para ello, se deben definir unas métricas de estudio del hardware usado, con tal de poder realizar una comparación posterior cuando el proyecto se ha dado por finalizado.

1.5 Preparación del entorno

Para el desarrollo del proyecto es necesario un entorno de desarrollo y simulación para poder comprobar los resultados. El entorno para este caso consiste en unas máquinas virtuales mediante la herramienta VMWare.

2. Ejecución

2.1 Containerización aplicaciones

Realización de la containerización de las aplicaciones mediante el uso de la herramienta Docker.

2.2 Desarrollo orquestador

Desarrollo de una infraestructura de automatización de la creación de los contenedores, así como de la posible migración de los mismos. Realización de la interfaz gráfica de este orquestador mediante una aplicación web.

3. Evaluación

3.1 Comprobación de los resultados de la ejecución

Evaluación de los resultados obtenidos durante la ejecución. Si estos resultados son incorrectos o insatisfactorios, se debe volver al punto anterior para seguir con el desarrollo. Éste es un problema muy susceptible de que suceda, por lo cual ya está contemplado en el cómputo del tiempo destinado en cada fase.

3.2 Definición de las métricas

Re-definición de las métricas de estudio escogidas en la fase inicial del proyecto. Previamente a esto se ha de estar completamente seguro de que el resultado de la evaluación es correcto.

3.3 Comparación con la fase inicial

Comparación de las métricas con las tomadas en la fase inicial. Se deben realizar gráficos para una mejor visión de los resultados. En caso de no estar conformes, aún se puede volver a la fase anterior para la realización de algunas mejoras.

4. Go-Live

4.1 Test de usuarios

Los usuarios realizan una comprobación de la funcionalidad del proyecto. En este caso, como el proyecto no va dirigido a los usuarios directamente, el testeo se realiza mediante problemas que puedan surgir de tener las aplicaciones contenerizadas o de la propia infraestructura de creación.

5. Memoria

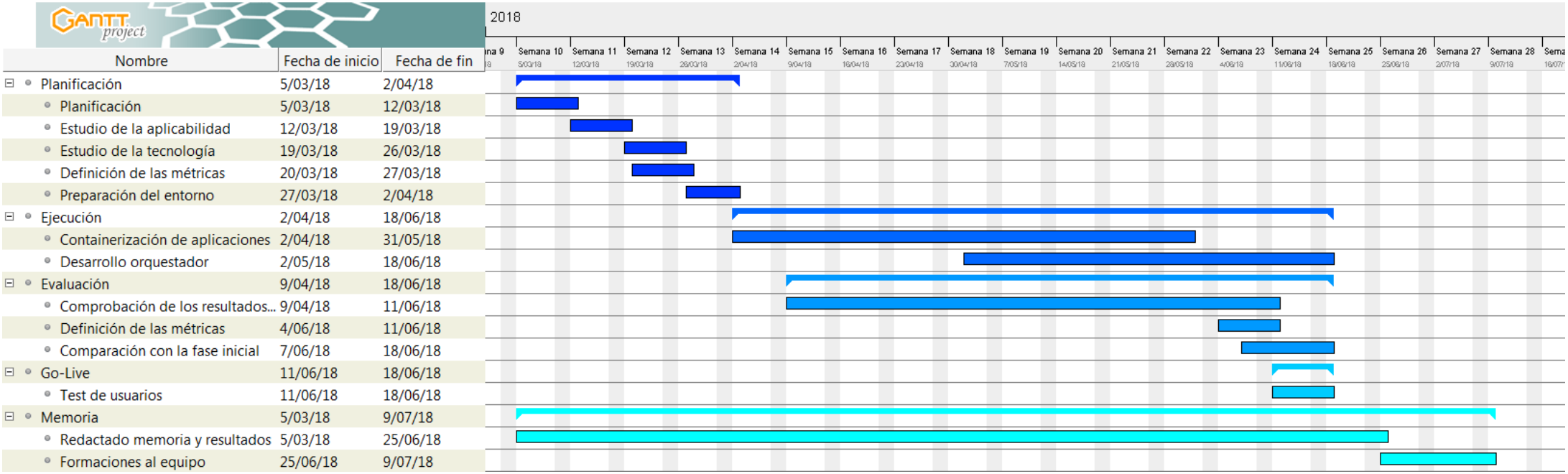
5.1 Redactado memoria y resultados

Últimas pinceladas del documento de la memoria, así como la realización de tablas y gráficos para una mejor interpretación de los resultados del estudio.

5.2 Formaciones al equipo

Por último, es importante formar al equipo en todo lo aprendido, ya que así el conocimiento se comparte y puede ser útil para la realización de futuros proyectos.

4.3. DIAGRAMA DE GANTT



Nombre	Fecha de inicio	Fecha de fin
Planificación	5/03/18	2/04/18
Planificación	5/03/18	12/03/18
Estudio de la aplicabilidad	12/03/18	19/03/18
Estudio de la tecnología	19/03/18	26/03/18
Definición de las métricas	20/03/18	27/03/18
Preparación del entorno	27/03/18	2/04/18
Ejecución	2/04/18	18/06/18
Containerización de aplicaciones	2/04/18	31/05/18
Desarrollo orquestador	2/05/18	18/06/18
Evaluación	9/04/18	18/06/18
Comprobación de los resultados...	9/04/18	11/06/18
Definición de las métricas	4/06/18	11/06/18
Comparación con la fase inicial	7/06/18	18/06/18
Go-Live	11/06/18	18/06/18
Test de usuarios	11/06/18	18/06/18
Memoria	5/03/18	9/07/18
Redactado memoria y resultados	5/03/18	25/06/18
Formaciones al equipo	25/06/18	9/07/18

Figura 2: Diagrama de Gantt

4.4. VALORACIÓN DE ALTERNATIVAS Y PLAN DE ACCIÓN

Durante el desarrollo del proyecto se pueden ocasionar imprevistos indirectos. Como la infraestructura de la empresa ya está montada, la resolución a estos imprevistos no requiere de una acción concreta y por tanto, no afectarían en gran medida a la resolución del proyecto. Los posibles imprevistos indirectos que se puedan ocasionar y su resolución se detallan a continuación:

- Fallo servidor físico: Las máquinas virtuales serán automáticamente migradas a otro servidor físico del clúster de producción.
- Fallo servidor virtual: Monitorización de servicios y generación de alerta. Migración de los contenedores a otra máquina.
- Caída eléctrica: El datacenter donde se alojan los servidores virtuales tiene clasificación TIER 3, con lo cual tiene garantizado un uptime de 99,982% anual mediante dos líneas de corriente que llegan al edificio por diferentes vías y provistas por diferentes compañías eléctricas.
- Caída Internet: Se dispone de una línea principal y backup en todas las sedes con un uptime garantizado de 99%.
- Host no admite Docker: Docker es compatible con la mayoría de Sistemas Operativos, pero en caso de que no fuera así, la empresa dispone de diversos Sistemas Operativos para instalar.
- Aplicación no Dockerizable: Es improbable debido a las pocas limitaciones de Docker, pero la empresa dispone de diversas aplicaciones web en tal caso.

5. Conocimientos

5.1. CONCEPTOS SOBRE CONTAINERIZACIÓN

5.1.1. Conceptos sobre contenedores

Los contenedores software [32; 33] son paquetes de elementos que permiten la ejecución de una determinada aplicación en cualquier Sistema Operativo. Para ello, el contenedor contiene, empaquetada, la imagen del Sistema Operativo para la cual se diseñó, junto con todas las dependencias que la aplicación en cuestión pueda necesitar para su funcionamiento y el código fuente de la misma. Es decir, dentro del contenedor se está ejecutando un Sistema Operativo que podría no ser el mismo que se ejecuta en la máquina host. En ese aspecto, se asemeja a la virtualización.

Volúmenes, ficheros compartidos, puertos, especificaciones hardware, etc, del Sistema Operativo se definen al crear el contenedor. Por tanto, los documentos estáticos como el código de la aplicación se empaquetan conjuntamente con la imagen y, los documentos dinámicos se han de guardar en el host local y copiarlos al contenedor o compartir las carpetas de dentro y fuera del contenedor. Para la realización de estas acciones, es decir, la gestión de la capa intermedia entre el contenedor y la máquina host, existen diversas aplicaciones. No obstante, para este proyecto se ha utilizado Docker, ya que es la herramienta que más auge está teniendo actualmente, con una gran comunidad de usuarios, una enorme variedad de imágenes y una alta fiabilidad, tal como demuestran las múltiples empresas que ya lo incorporan en sus sistemas.

5.1.2. Conceptos sobre Docker

5.1.2.1. Conceptos básicos

Docker es un proyecto de código libre que permite el despliegue y gestión de aplicaciones dentro de contenedores software. Este software proporciona una capa adicional de abstracción y automatización de la virtualización. Es decir, es el software que gestiona la capa intermedia entre el contenedor y la máquina host. Docker se encarga de la repartición de recursos entre la máquina host y los contenedores, del aislamiento del espacio de memoria, de la gestión de las interfaces de red, etc.

5.1.2.2. Diferencias entre Docker y una VM

La diferencia más clara entre Docker y una Máquina Virtual [34] es la portabilidad de las aplicaciones. Con Docker, basta con encapsular la aplicación en un contenedor una única vez y funcionará en cualquier otro entorno. Sin embargo, una aplicación que se ejecute en una máquina virtual puede no ser compatible en otra máquina virtual debido a una distinta instalación del sistema operativo, entre otras posibles causas.

Sin embargo, la diferencia más importante que se pretende estudiar en este proyecto es la del uso de recursos. La infraestructura propuesta en este trabajo pretende explotar al máximo esta oportunidad de reducción de recursos.

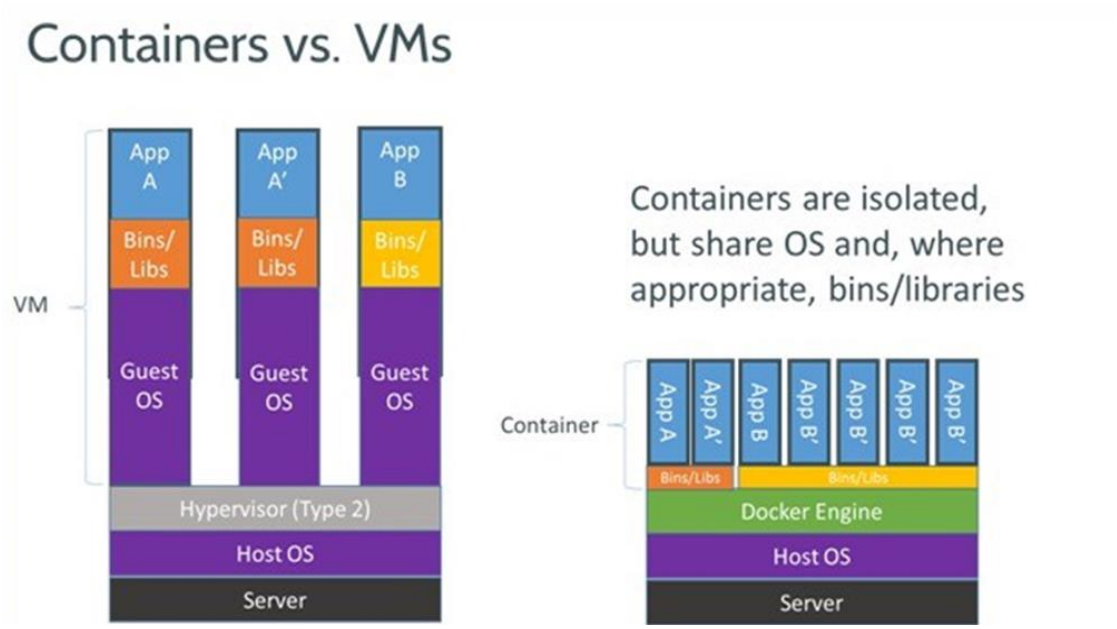


Figura 3: Comparación entre una infraestructura con VM y una con Contenedores [34]

5.1.2.3. Herramientas de Docker

Docker provee de diversas herramientas para el uso de su software. A continuación, se detallan las que han sido de mayor utilidad para el desarrollo de este proyecto.

Dockerfile (35)

Dockerfile es una herramienta propia de Docker que permite la redefinición de imágenes de sistemas operativos para el uso en contenedores. Mediante un fichero de texto llamado Dockerfile se define la imagen que se va a tomar como base y se añaden los comandos de instalación tanto las aplicaciones como los módulos que esa imagen requiere tener para el uso de la aplicación que se pretende contenerizar, entre otras muchas funcionalidades distintas. En otras palabras, un fichero Dockerfile es un script similar a un Makefile que monta la imagen mediante el comando *docker build.* Esto permite y facilita la automatización de la creación de imágenes.

Entre las múltiples instrucciones que Dockerfile pone a disposición, destacan:

- RUN, que permite la ejecución de instrucciones por línea de comandos en la imagen que se está montando.
- ADD, que permite el copiado de ficheros de la máquina host a la imagen que se está montando.
- WORKDIR, que cambia el directorio en el cual se ejecutan las anteriores instrucciones (Similar a un cd).
- USER, que añade usuarios a la imagen.
- EXPOSE, que permite que la imagen escuche instrucciones de un puerto de red especificado.

Docker-compose (36)

Docker compose es una herramienta ideada para definir contenedores. Mediante un fichero tipo YAML[2] bautizado como docker-compose, se selecciona la imagen que se va a usar, el nombre del contenedor, las interfaces de red, compartición de ficheros, los servicios de la aplicación y se limitan los recursos del Sistema Operativo, entre muchas otras funcionalidades. Una vez creado, mediante el comando *docker-compose up* el contenedor arrancará con la imagen establecida y los parámetros definidos. En caso de no disponer de la imagen, ésta se descarga

[2] Formato de serialización de datos similar a XML.

automáticamente del repositorio de Docker o montará la definida por el fichero Dockerfile detallado en el punto anterior. Una vez el contenedor está en marcha, se puede trabajar con él con normalidad.

Docker compose facilita las siguientes funcionalidades:

- Múltiples entornos aislados en un único host: Con docker compose puedes disponer de diversos entornos para, por ejemplo, realizar pruebas en desarrollo.
- Preserva los datos de los contenedores: Como docker-compose define un mapeo de carpetas con la máquina host, al levantar y apagar los contenedores no existe pérdida de datos.
- Solo crea contenedores que han cambiado: Es decir, si realizas algún cambio en tu docker-compose.yml, al ejecutar el comando docker-compose up se creará otro contenedor nuevo. De otro modo, seguirá siendo el mismo.

Sintetizando, trabajar con docker compose simplifica muchísimo la creación de los contenedores. Tan solo se deben ejecutar tres pasos:

1. Definir el fichero *Dockerfile* con el cual se generará la imagen.
2. Definir los servicios, limites, ficheros, etc que requiere la aplicación mediante el fichero *docker-compose.yml.*
3. Levantar el contenedor con el comando *docker-compose up.*

Docker API [37]

Por último, Docker dispone de una API tanto en lenguaje Python como en GO para permitir a los desarrolladores trabajar con su software. Dicha herramienta ha sido empleada para el desarrollo del orquestador propuesto en este proyecto. Pese a existir API para casi cualquier lenguaje, Docker solo provee de forma oficial la de estos dos lenguajes, por tanto, por nociones previas en él, se ha escogido Python como lenguaje de programación del orquestador.

5.2. INTEGRACIÓN DE LOS CONOCIMIENTOS

Para la realización de esta estructura propuesta se han aplicado los conocimientos aprendidos en asignaturas de sistemas operativos, CPD y redes.

Para la containerización de las aplicaciones se ha tenido que aprender funcionalidades de una herramienta nueva, Docker, pero también se han requerido conocimientos ya adquiridos en otras disciplinas impartidas en la universidad, tales como conocimientos en sistemas operativos, redes y software.

Para realizar una interfaz más amigable con el orquestador, se propone, además de la implementación de dicho orquestador, realizar una página web en la cual ver el estado de los contenedores y poder realizar un control de los mismos. El orquestador y la web se comunican mediante unas tablas SQL, en las cuales la web envía las órdenes y el orquestador las procesa. Las funcionalidades que se envían al orquestador desde la web son parada, arranque y migración de contenedores de un nodo a otro bajo demanda, creación y eliminado de contenedores de imágenes ya creadas, monitorización del tráfico, hardware, etc. Sin embargo, el orquestador también dispondrá de autonomía para migrar contenedores en caso de caída del nodo o saturación del tráfico, memoria, etc en la máquina host. También generará alertas en caso de fallo con alguno de los contenedores y/o nodo, mostrándolo vía web.

El código del orquestador se realiza en lenguaje Python, ya que, como ya se ha comentado, la API oficial de Docker sólo está para este lenguaje. Este lenguaje ya es familiar debido a los conocimientos aprendidos en las asignaturas de DSBM y ROB. No obstante, para el desarrollo de la página web, se han tenido que aprender lenguajes como HTML y PHP. Para la comunicación con la base de datos, se han aplicado los conocimientos aprendidos en las asignaturas de BD.

5.3. IDENTIFICACIÓN DE LEYES Y REGULACIONES

El desarrollo de este proyecto no está sujeto a ninguna ley o normativa. El proyecto modifica la infraestructura de la empresa, y no las webs en sí, por tanto, al tratarse de aplicaciones web que la empresa ya disponía, éstas ya están reguladas bajo las normativas correspondientes, tales como la GDPR, las cookies, las condiciones de uso, etc. según el caso del cual se trate.

Sin embargo, es importante tener en cuenta la normativa del código deontológico del colegio de informáticos[6] a la hora de realizar un proyecto de esta categoría, ya que recoge una serie de normas de carácter ético que son de recomendado cumplimiento.

6. Análisis de la sostenibilidad

6.1. AUTOEVALUACIÓN DE LA SOSTENIBILIDAD

Mediante la realización de un cuestionario [38] se realiza una autoevaluación de los conocimientos sobre sostenibilidad y desarrollo social. Dicho cuestionario requiere responder, de forma honesta, unas preguntas relacionadas con la sostenibilidad social, medioambiental y económica.

El cuestionario abarca dos ítems principales. El conocimiento personal sobre dichos conceptos y cómo aplican para el desarrollo de proyectos.

En cuanto a conocimiento sobre este tema, se han hecho diversas conferencias en el ámbito universitario sobre sostenibilidad medioambiental. Gracias a esto, conozco en mejor medida el impacto ambiental de las TIC en el planeta y las nuevas propuestas de reutilización de los recursos. Pese a eso, aún son algunos conceptos los que desconocía, como por ejemplo cuáles son los indicadores utilizados para medir el impacto ambiental de un proyecto.

En cuanto a sostenibilidad social, soy muy consciente la problemática de la inequidad social que existe actualmente, así como de otros problemas relacionados con la diversidad y la transparencia. También gracias a la universidad, soy capaz de desarrollar trabajos colaborativos, ya que la mayoría de proyectos que he realizado se han realizado en equipo. Aun así, desconocía algunos conceptos, tales como justicia social o la ergonomía de los productos TIC.

Por último, sobre sostenibilidad económica gracias a que en la universidad se realiza de forma obligatoria una asignatura sobre economía, y, pese a ser sólo en un curso de un semestre y por tanto el contenido que se trata es relativamente escueto, conocía algunos conceptos, como los costes fijos, los variables y las amortizaciones. Por otra parte, desconocía cómo sería la realización de la gestión económica de un proyecto. Gracias a este proyecto y a la ayuda recibida por parte de la empresa, puedo llegar a comprender un poco más dicho concepto.

Concluyendo, considero importante el conocimiento de la problemática social y medioambiental de la tecnología actual y creo que es algo que se debería tener en cuenta siempre. Una sociedad más justa permite un mejor desarrollo de la misma, ya que la aportación de todos hace que el desarrollo de un proyecto en este caso disponga de más puntos de vista y, por tanto, mejores opciones de éxito.

En cuanto a la problemática medioambiental, solamente disponemos de un planeta y es importante cuidarlo para nuestra propia salud y bienestar.

6.2. ANÁLISIS DE LA SOSTENIBILIDAD

Observando la matriz de sostenibilidad proporcionada se observa que el Trabajo de Final de Grado abarca un pequeño tramo de la sostenibilidad de todo el ciclo de vida de un proyecto. Dicho tramo es vital, ya que está incluido en la parte de planificación y desarrollo del proyecto. Se deben planificar y estudiar todos los aspectos de las distintas dimensiones detalladas en los siguientes apartados con tal de tenerlos en cuenta durante desarrollo del proyecto.

6.3. DIMENSIÓN ECONÓMICA

Mediante la matriz de sostenibilidad proporcionada, se analizan diversos aspectos en cuanto a la dimensión económica del proyecto.

6.3.1. Presupuesto

Para la planificación de un proyecto se debe llevar a cabo una estimación del presupuesto que se va a invertir en el mismo, con tal de estudiar si el proyecto es viable económicamente. Para ello intervienen los siguientes aspectos:

6.3.1.1. Costes Directos

Los costes directos se dividen en costes humanos y costes de material, tanto Hardware como Software.

6.3.1.1.1 Costes Humanos

Identificación	*Dedicación*	*Precio/hora*	*Total*
Director	20h	50€	1000€
DevOps	40h	50€	2000€
Desarrollador Horario laboral	200h	10€	2000€
Desarrollador Horario no laboral	250h	0€	0€

Figura 4: Tabla con los costes humanos derivados del proyecto

La dedicación esperada por parte del desarrollador se ha visto incrementada en cuanto a horas invertidas respecto de lo que inicialmente se planeaba debido a complicaciones derivadas de la implementación del orquestador. Esta dedicación se ha realizado fuera de horario laboral, por tanto, no ha añadido ningún extra al coste humano del proyecto calculado inicialmente.

6.3.1.1.2 Costes Materiales

6.3.1.1.2.1 Recursos Hardware

Identificación	*Estimación precio*	*Vida útil*	*Amortización*	*Descripción*
Servidores Físicos	0€	5a	0	Servidores donde correrá el proyecto.
Deploy Servidores Virtuales	1040€	-	-	Servidor virtual donde se aloja.
Infraestructura de red	0€	5a	0	Conexión local e internet.
Hosting equipo en datacenter	0€	-	-	Alquiler de equipos al datacenter.

Figura 5: Tabla con los costes de recursos hardware del proyecto.

6.3.1.1.2.2 Recursos Software

Identificación	*Estimación precio*	*Vida útil*	*Amortización*
Docker	0€	-	-
Google drive	0€	-	-
Putty	0€	-	-
VMWare	0€	-	-
WinSCP	0€	-	-
Sonic Wall	0€	-	-
Trello	0€	-	-
VCenter	0€	-	-
Lenguajes de programación	0€	-	-
Licencias SO	0€	-	-

Figura 6: Tabla con los costes de recursos software del proyecto.

6.3.1.2 Costes Indirectos

Identificación	*Estimación precio*	*Vida útil*	*Amortización*	*Descripción*
Mantenimiento infraestructura de red	0€	-	-	Pago para el mantenimiento de la red.
Mantenimiento cluster VMWare	0€	-	-	Pago para el mantenimiento del cluster.
Servicio WAN	0€	-	-	Mantenimiento red local
Gastos oficina	50€	5m	10€/m	Material de oficina
Gastos equipos	800€	5a	160€/a	Ordenador portátil

Figura 7: Tabla con los costes indirectos del proyecto.

6.3.1.3 Contingencias e Imprevistos

Existen una serie de imprevistos que pueden surgir a la hora de desarrollar un proyecto. En temas económicos, estos imprevistos pueden ocasionar grandes pérdidas. Para evitar esto, a continuación se detalla el plan de acción para cada contingencia:

- Fallo servidor físico: Las máquinas virtuales serán automáticamente migradas a otro servidor físico del clúster de producción gracias a la configuración HA actual del mismo.
- Fallo servidor virtual: Monitorización de servicios y generación de alerta al servicio de soporte. SLA 8h. Migración de contenedores
- Caída eléctrica: El datacenter donde se alojan los servidores virtuales tiene clasificación TIER 3, con lo cual tiene garantizado un uptime de 99,982% anual mediante dos líneas de corriente que llegan al edificio por diferentes vías y provistas por diferentes compañías eléctricas. Todo equipo en el datacenter está conectado a ambas líneas.
- Caída Internet: Se dispone de una línea principal y backup en todas las sedes con un uptime garantizado de 99% anual.

6.3.1.4 Coste total

Identificación	*Total*
Costes Humanos	5000€
Costes Hardware	1040€
Costes Software	0€
Costes Indirectos	850€
Total	**6890€**

Figura 8: Tabla resumen con el coste total del proyecto.

Como se observa en la tabla anterior, el coste total aproximado del proyecto es de 6890€. Como se deriva de los datos, la mayor parte de este coste es debido a las horas invertidas por parte de los trabajadores, pues como se ha comentado ya, no es necesaria la compra de ningún componente nuevo, ya que se usa la infraestructura disponible de la empresa. Todos estos datos se han extraído de la información aportada por los responsables de la empresa.

6.3.2. Reflexión

Debido a que la infraestructura de la empresa ya está montada, el coste de la puesta en producción es sin duda bajo, ya que, al trabajar sobre máquinas virtuales, éstas son desplegables en cualquier servidor contando el coste de la máquina virtual tan solo. Si se deben tener en cuenta los costes downtime que generaría este proyecto, puesto que algunas de las aplicaciones de la empresa dependen de forma directa de él.

6.4. DIMENSIÓN AMBIENTAL

Mediante la matriz de sostenibilidad proporcionada, se realiza una reflexión sobre la dimensión ambiental del proyecto.

6.4.1 Reflexión

El impacto ambiental que se deriva directamente de este proyecto es nulo, puesto que no se añade ningún tipo de componente nuevo para la empresa que pueda generar dicho impacto, ya que se trabaja mediante virtualización. Sin embargo, el impacto ambiental que se deriva de forma indirecta es debido al consumo que puedan generar los servidores a causa del aumento de carga de trabajo que se obtiene de la puesta en producción de este proyecto junto con todos los otros que ya estaban.

En su fase final, el proyecto, de forma indirecta, minimiza el impacto ambiental indirecto, gracias a que, al unificar unas cuantas aplicaciones mediante la virtualización de las mismas, el uso de servidores se minimiza, debido a que ya no es necesario tener máquinas virtuales dedicadas para cada aplicación.

Esta idea de reducción de costes, tanto económicos como ambientales, ya se lleva a cabo en diversos lugares, gracias a que la amplia comunidad de Docker dispone de muchísimas aplicaciones que utilizan gran cantidad de empresas. Al no disponer de esta solución en la empresa en la que trabajo, ésta sería una gran mejora.

6.5. DIMENSIÓN SOCIAL

Mediante la matriz de sostenibilidad proporcionada, se realiza una reflexión sobre la dimensión social del proyecto.

6.5.1 Reflexión

La realización de este proyecto, a nivel personal, aporta la satisfacción de aprender a trabajar con una herramienta totalmente nueva para mí. El proyecto es un gran primer paso para esto y para una posterior certificación en Docker en caso de que la experiencia resulte satisfactoria. También gracias a realizar el proyecto en una empresa he aprendido los métodos de realización de proyectos en el entorno profesional.

Este proyecto facilita la tarea de la administración de los sistemas gracias a que, como ya se ha explicado, unifica la localización de las aplicaciones, reduciendo el uso de servidores y facilitando su gestión.

La empresa requería una necesidad del proyecto, debido a que, se destinan muchos recursos dedicados únicamente a una aplicación, tales como puertos, máquinas virtuales, servidores, etc. Además de esto, permite también el estudio de la virtualización, realizando comparaciones en cuanto a rendimientos y usos de memoria por parte de los servidores, entre otros aspectos.

6.6. MATRIZ DE SOSTENIBILIDAD

Para resumir y sintetizar todos los puntos explicados anteriormente se realiza una matriz de sostenibilidad [39]:

Dimensiones	*PPP*	*Vida útil*	*Riesgos*
Económica	Menor coste que antes debido a la reducción de máquinas/ servidores.	Hasta que dicha tecnología quede obsoleta.	No supone ningún riesgo económico.
Ambiental	Menor consumo que antes debido a la reducción de máquinas/servidores.	Su vida útil no varía su impacto ecológico.	No supone ningún riesgo ambiental.
Social	Satisfacción de aprender y desarrollar una nueva tecnología.	Mejora las tarea de los administradores de sistemas.	No supone ningún riesgo social.

Figura 9: Matriz de sostenibilidad que se extrae del proyecto.

Con todos los motivos detallados en las distintas dimensiones, podemos afirmar que se trata de un proyecto sostenible.

7. Diseño

7.1. DETALLES DE LA INFRAESTRUCTURA

7.1.1. Descripción de la infraestructura diseñada

Actualmente, la infraestructura de la empresa se compone de una máquina virtual para una o dos aplicaciones web, conectadas a una base de datos y con una máquina ejecutando un proxy para dirigir el tráfico de red.

En el esquema siguiente se muestra la nueva propuesta de infraestructura desarrollada en este proyecto y se detalla cada uno de los componentes.

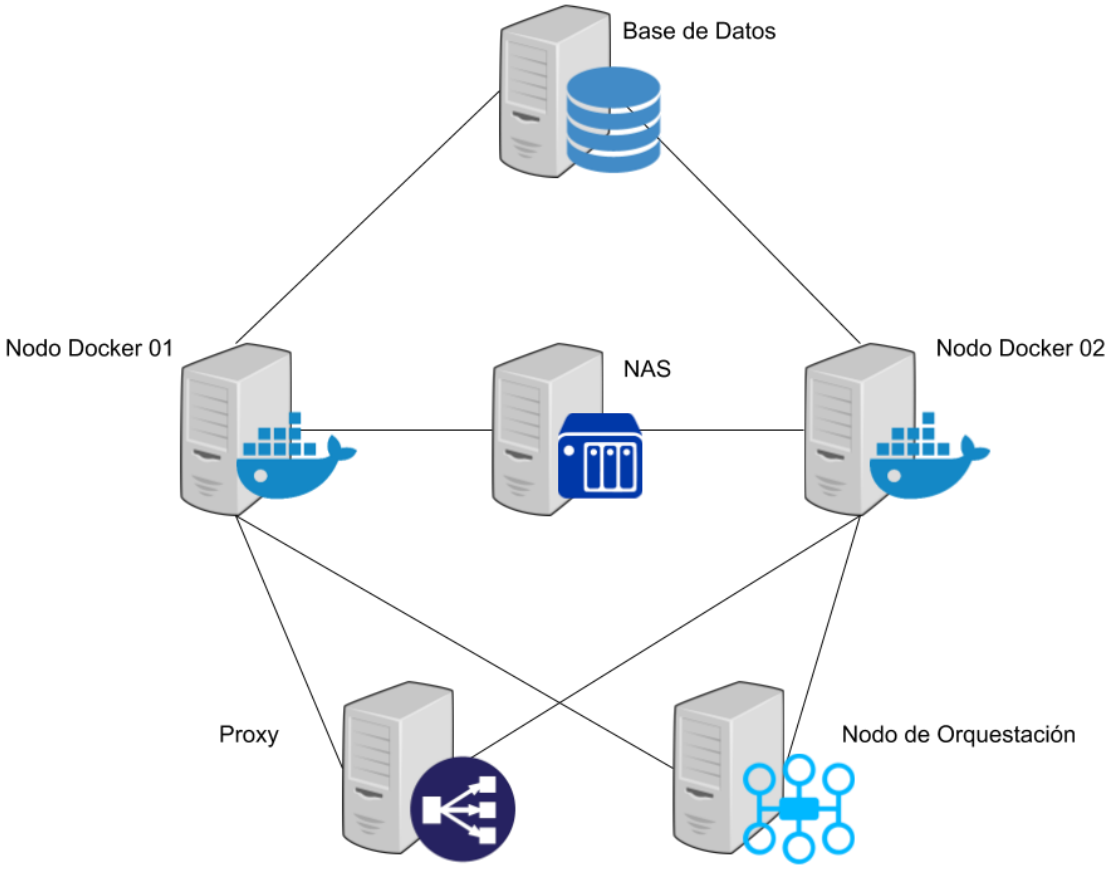


Figura 10: Esquema de las conexiones de los elementos del nuevo modelo de infraestructura.

Como se puede observar en la figura, la infraestructura propuesta para este proyecto consta de 6 máquinas implicadas. Todas ellas son máquinas virtuales en el mismo servidor físico, a excepción del NAS, que es una máquina exclusiva.

Nodos Docker

Estas máquinas virtuales son las encargadas de funcionar como hosts de los contenedores Docker. El hecho de que existan es debido a que permiten realizar migraciones de contenedores en caso de caída del nodo, saturación, o bajo demanda. Ambas son idénticas en cuanto a hardware y configuraciones del sistema. A continuación se detalla dicho hardware:

Nodos de Docker + Nodo Orquestador	
Arquitectura	x86, 64 bits
Nº CPUs	4
Frecuencia	2194Mhz
Niveles de Cachés	3
Tamaño de las Cachés	32Kb, 256Kb y 20480Kb
RAM	6Gb
Swap	2Gb
Disco	50Gb

Figura 11: Resumen del hardware de los nodos de Docker y Orquestación.

Nodo de orquestación

Esta máquina virtual ejecuta el programa del orquestador y aloja la aplicación web que hace de interfaz gráfica del mismo. Se encarga de comprobar el estado y enviar las órdenes a los nodos de producción. Su hardware es idéntico al de los nodos Docker.

Nodo NAS

Máquina física en la cual se encuentran los archivos necesarios para la configuración de las imágenes y el despliegue de los contenedores. Ambos nodos de producción tienen acceso mediante una carpeta compartida, lo cual facilita las migraciones y hace que la infraestructura sea escalable. Compartida con el resto de aplicaciones de la empresa. Éstas son sus características:

NAS	
Modelo	EMC VNX5300
Nº Discos	9
Capacidad Discos	538GB y 1834GB
Nº LUNS	5
Capacidad LUNS	1Tb cada una y otra de 2Tb
RAID	5

Figura 12: Resumen del hardware del NAS.

Nodo Bases de datos

Máquina virtual de Base de Datos, la cual contiene los datos que las aplicaciones web almacenan durante su funcionamiento. Compartida con el resto de aplicaciones de la empresa. Su hardware se detalla a continuación:

Base de Datos	
Arquitectura	x86, 64 bits
Nº CPUs	4
Frecuencia	2194 MHz
Niveles de Cachés	3
Tamaño de las Cachés	32Kb, 256Kb, 20480Kb
RAM	4Gb
Swap	4Gb

Figura 13: Resumen del hardware de la máquina de base de datos.

Nodo Proxy

Máquina virtual encargada de ejecutar el rol de servidor proxy de la infraestructura. Su uso está destinado a la redirección del tráfico de red según donde se localice el contenedor que ejecuta la aplicación a la cual se pretende acceder. Si ese contenedor cae y es migrado, este servidor se encarga de redirigir las peticiones automáticamente. Compartido con el resto de aplicaciones de la empresa. Su hardware se muestra a continuación:

Proxy	
Arquitectura	x86, 64 bits
Nº CPUs	1
Frecuencia	2194 MHz
Niveles de Cachés	3
Tamaño de las Cachés	32Kb, 256Kb, 20480Kb
RAM	2Gb
Swap	2Gb

Figura 14: Resumen del hardware de la máquina proxy.

7.1.2. Descripción de las tecnologías potenciales

Para el desarrollo del proyecto se ha utilizado la tecnología de la cual dispone la empresa. Existen mejores máquinas, pero para la infraestructura que se ha desarrollado, el hardware disponible ya es más que suficiente.

La versión de VMware que se utiliza es la VMware Hypervisor 6.5. Actualmente, ésta es la última versión estable que ha salido al mercado de VMware Hypervisor.

El sistema operativo del que dispone la máquina es un CentOS 7.4. Actualmente, la ésta es la última versión disponible de CentOS, puesto que garantizan soporte para cada versión durante 10 años.

Por último, ya que Docker es un software libre y de nueva instalación para la empresa, para el desarrollo del proyecto se utiliza la última versión estable lanzada, es decir la 18.

7.2. DISEÑO DEL ORQUESTADOR

Para explotar al máximo las ventajas de la containerización, se pretende desarrollar un orquestador de contenedores para la infraestructura diseñada en este proyecto. Para que sea accesible desde cualquier lugar, se decide que su interfaz gráfica consistirá en una página web.

El orquestador debe realizar las siguientes funcionalidades:

- Listado de contenedores activos por nodo: La interfaz web debe permitir poder ver el estado de todos los contenedores de forma clara y entendible, con tal que no sea un problema a gran escala.
- Arranque y parada de contenedores bajo demanda: Para cada contenedor, se debe permitir una gestión del mismo. Para ello, interfaz y programa deben estar conectados en todo momento para poder resolver las peticiones.
- Migración bajo demanda y automática en caso de posible caída del nodo o saturación del mismo: Una de las ventajas que ofrecen los contenedores es la migración. Con tal de aprovechar dicha ventaja, el orquestador deberá permitir al usuario migrar los contenedores de nodo cuando lo necesite. Incluso, el orquestador debe ser capaz de, si existe saturación o caída de un nodo, realizar una migración de forma automática, para garantizar que las aplicaciones web están constantemente arriba.
- Monitorización, gráficos e información detallada de cada contenedor: Para poder facilitar el control de los administradores de sistemas, el orquestador deberá mostrar las estadísticas y detalles de cada contenedor por la interfaz gráfica para facilitar la monitorización de los contenedores.
- Monitorización y gráficos de los nodos: Por el mismo motivo que el punto anterior, se deben monitorizar el estado y el uso de recursos que se está haciendo en cada nodo de la nueva infraestructura, con tal de conocer el estado del servicio en todo momento.

8. Implementación

8.1. ENTORNO DE DESARROLLO

El entorno utilizado para el desarrollo es una pequeña máquina virtual con un sistema operativo CentOS 7 corriendo en ella. No dispone de interfaz gráfica, por tanto, todo se realiza desde línea de comandos mediante SSH. La máquina utilizada para el desarrollo es idéntica a la que se utiliza en los nodos de producción. A continuación, se detallan los pasos para la instalación de los nodos de producción.

8.2. INSTALACIONES Y CONFIGURACIONES INICIALES

8.2.1. Creación de la máquina virtual

Primeramente, se debe realizar la creación de las máquinas virtuales. Para este proyecto se han creado las máquinas desde la herramienta VCenter con las características hardware definidas previamente, pero para la implementación de esta infraestructura no es requerido ningún hardware especial.

Para esta infraestructura requerimos de tres máquinas, dos nodos de producción y otro para la ejecución del orquestador. Ambas máquinas de producción son idénticas, por tanto, VCenter permite la creación de una como copia de la otra, con lo que sólo es necesario crear una desde cero.

Los detalles de la creación de las máquinas virtuales se omiten puesto que, como ya se ha dicho, es irrelevante para la implementación de la infraestructura propuesta ya que es compatible con todo tipo de hardware.

8.2.2. Instalación de CentOS7

Lo mismo ocurre con el sistema Operativo. Para este proyecto se ha utilizado un CentOS 7, pero cualquier Sistema (preferiblemente Linux) con cualquier tipo de configuración podría ser susceptible de uso para la implementación de la infraestructura propuesta, por tanto, también se omiten dichos pasos.

8.2.3. Instalación del NAS en la infraestructura

Para que la compartición de ficheros de configuración y datos entre los nodos sea lo más fácil y escalable posible, dichos datos se guardan en dos máquinas distintas. Los ficheros de configuración se almacenan en un NAS montado en los nodos. Dichos ficheros son el Dockerfile, el Docker-compose y demás ficheros de configuración necesarios para la creación de las imágenes y los contenedores.

Por otro lado, los datos que requieren y almacenan las aplicaciones se guardan en una máquina mySQL. MySQL ya permite el acceso remoto a sus bases de datos, por tanto, la lectura y escritura de los datos en las bases de datos se realiza de forma fácil simplemente añadiendo el módulo de mysql a las imágenes en el Dockerfile. Esto garantiza también que no hay pérdida de datos entre los dos nodos de producción durante caídas y migraciones.

8.2.4. Instalación de Docker

Para la instalación de la CLI de Docker, como los nodos ejecutan un CentOS 7, se debe ejecutar el comando:

- *yum install docker.*

Una vez instalado, es preciso levantar el daemon de docker. Para ello se requiere el comando:

- *systemctl start docker*.

Para hacer que el daemon se levante con cada reinicio, es recomendable activar dicha funcionalidad mediante el comando:

- *systemctl enable docker.*

Para comprobar que efectivamente el daemon está arriba, se puede realizar un *docker ps* para listar los contenedores.

8.2.5. Instalación de Docker-compose

El siguiente paso es instalar la herramienta docker-compose. Esta herramienta también se encarga del montaje de las imágenes mediante el documento Dockerfile, la cual ya viene conjuntamente con la instalación de Docker. Como se utiliza un CentOS para este proyecto, primeramente es preciso instalar la herramienta *pip-python*. Para ello se ejecuta:

- *yum install epel-release* y *yum install -y python-pip*.

Ahora ya es posible instalar docker-compose mediante el comando:

- *pip install docker-compose*

Por último, para asegurar que docker compose funciona con normalidad, es recomendable ejecutar el comando:

- *yum upgrade python**.

Para comprobar que docker-compose se ha instalado correctamente, se puede realizar un *docker-compose -v*, el cual nos dará la versión del mismo.

8.2.6. Instalación del segundo nodo de producción

Con todo esto instalado se realiza un clonado de la máquina virtual. Esto hace que no se deba volver a instalar todo de nuevo y, además, asegura que la instalación de los programas será la misma en ambos nodos. La herramienta VCenter dispone de dicha funcionalidad, pero cualquier otra herramienta que permita realizar dicha función sería válida para esta acción.

8.2.7. Configuración del host remoto de Docker

A continuación, es necesario conseguir que los nodos host de Docker escuchen las órdenes que les llegan por un puerto en concreto.

Para hacer que el daemon de docker escuche lo que entra por el puerto *2375* (puerto reservado para docker). Esto se consigue mediante el comando:

- *dockerd -H tcp://0.0.0.0:2375*

Previo a esto, se debe asegurar que el servicio de docker está parado:

- *systemctl stop docker*

Para comprobar que el daemon está corriendo, se pueden visualizar los procesos que están escuchando órdenes de puertos mediante el comando:

- *netstat -tulnp*.

Para que estos cambios sean permanentes en una posible caída del nodo, se debe configurar este puerto por defecto en el fichero de configuración *daemon.json* que se encuentra en la ruta */etc/docker*. Simplemente es necesario añadir la siguiente línea entre las llaves *{}* al fichero anterior:

- *"hosts": ["tcp://IP:2375"]*

donde IP es la dirección IP de la máquina en la que se encuentra el host de Docker. Para comprobar que estos cambios son efectivos, es recomendable realizar un *systemctl enable docker* para que refresque la nueva configuración.

Si ahora se quisiera utilizar Docker con normalidad no sería posible, puesto que el sistema indica que no encuentra el daemon de Docker. Para hacer que docker siga siendo usable en el mismo servidor en el que se está trabajando, se debe definir la variable global *DOCKER_HOST* para que apunte a la nueva dirección por la que el daemon recibe las órdenes, es decir, para que apunte a sí mismo.

Para ello, en este caso basta con realizar el comando:

- *export DOCKER_HOST=IP:2375*

donde IP es la dirección IP de la máquina en la que se encuentra el host de Docker. Sin embargo, esto sólo funciona para la sesión actual del usuario. Para hacer estos cambios permanentes para un mismo usuario de la máquina en diversos inicios de sesión, se debe ejecutar el comando:

- *echo 'export DOCKER_HOST=IP:2375' >> $HOME/.bashrc*

el cual copiará este export al script ubicado en $HOME/.bashrc, el cual se ejecuta en el arranque del usuario. Para comprobar que todo funciona correctamente, basta con cerrar la sesión y volver a entrar a la máquina.

Por último, debido a incompatibilidades con la versión de Docker actual y el sistema operativo CentOS 7, se debe crear un soft link de los ejecutables *docker-proxy-current* y *docker-runc-current* ubicados en la ruta */usr/libexec/docker/* mediante los comandos:

- *ln -s docker-runc-current docker-runc*
- *ln -s docker-proxy-current docker-proxy*

Una vez hecho esto, se deben copiar a alguno de los directorios incluidos en la variable *$PATH*. Esto se puede hacer con el comando:

- *cp docker-proxy /usr/local/bin/*
- *cp docker-runc /usr/local/bin/*

Con esto, ya es posible utilizar Docker con normalidad dentro del nodo de producción.

8.2.8. Comunicación entre nodos

Para que los nodos de Docker sean accesibles desde el nodo de orquestación, se debe ejecutar el comando en el nodo host de Docker:

- *firewall-cmd --permanent --zone=trusted --add-source=IP*

donde IP es la IP del nodo de orquestación. Para que estos cambios sean permanentes, basta con ejecutar:

- *firewall-cmd --reload*

con tal de que refresque la tabla de IPs.

Otro tipo de comunicación que el orquestador requiere es una comunicación mediante ssh. Para que esta conexión sea automatizada y segura, se deben definir los hosts como seguros. Para ello el nodo de orquestación debe generar una clave ssh mediante el comando:

- *ssh-keygen -t rsa -b 2048*

la cual compartirá con los otros dos nodos con el comando:

- ssh-copy-id IP

donde IP es la IP de los otros dos nodos de producción, para permitir la comunicación entre ambos de forma segura.

8.2.9. Configuración HA Proxy

Para realizar las redirecciones de los accesos a las aplicaciones web se ha usado la herramienta HA Proxy. Ésta comprueba que las IP's indicadas respondan y, en caso que una de ellas no, dirija los accesos a la otra. Para conseguir dicha finalidad existen diversas herramientas, por lo tanto, también se omitirá dicha configuración.

8.2.10. Instalación de la API de Docker

Por último, para realizar la orquestación de los contenedores, Docker provee una API oficial de comunicación. Para instalarla, se requiere la aplicación pip. Para instalar dicha aplicación una de las formas es utilizando el comando:

- *curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"*

A continuación, se ejecuta el script recién descargado con:

- python get-pip.py

Para comprobar que la actualización ha funcionado correctamente, se puede realizar una prueba utilizando el comando *pip --help.*

Finalmente, para instalar la API de Docker, basta con realizar:

- pip install docker

8.3. CONTAINERIZACIÓN DE APLICACIONES

Para los primeros pasos con esta nueva tecnología, se realizan las containerizaciones de 3 aplicaciones web muy simples, en contenedores que ejecutan apache con versiones de php 5.4, 7 y 7.2. Estas aplicaciones han servido para familiarizarse con el entorno y la tecnología y realizar las primeras pruebas.

Por parte de la empresa, a modo de inicio en este nuevo tipo de infraestructura, se proveen 3 aplicaciones web ya existentes para contenerizar. La containerización de dichas aplicaciones se ha realizado utilizando como imagen base las imágenes ya preparadas de apache con php definidas en la etapa de familiarización, pero añadiendo al fichero Dockerfile los módulos de php necesarios. No obstante, una de ellas se ha tenido que utilizar una imagen del sistema operativo de CentOS7 desde cero y se le han ido añadiendo las dependencias, módulos y configuraciones de ficheros mediante los ficheros Dockerfile y docker-compose. Además, se han tenido que configurar permisos para hacerlo accesible desde fuera del contenedor.

Por último, la empresa propone una cuarta aplicación para implementar directamente sobre docker. Se trata de una aplicación en Joomla, y por tanto simplemente con descargarse el repositorio oficial de la misma y ejecutar el contenedor ya se puede empezar a trabajar en ella sin ningún tipo de problema.

8.4. IMPLEMENTACIÓN DEL ORQUESTADOR

A continuación, se detalla en pseudocódigo la implementación del orquestador.

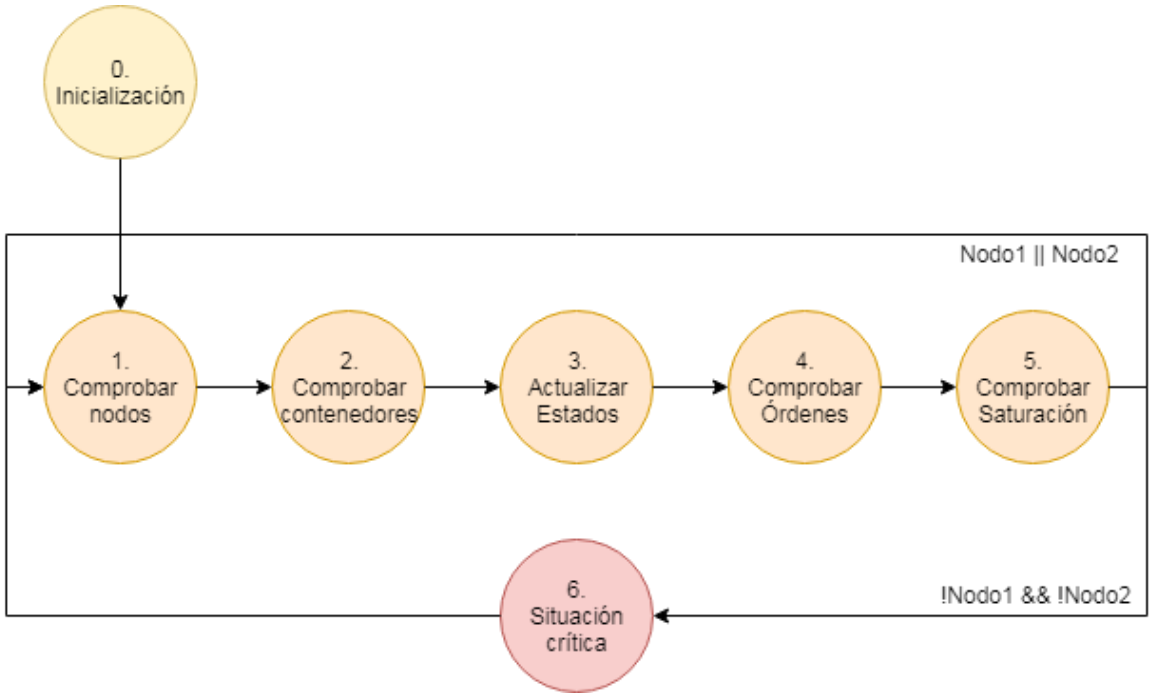


Figura 15: Esquema de estados del orquestador.

Tal y como se muestra en la figura, la ejecución del orquestador se compone de 7 grandes fases.

8.4.1. Fase Inicial: Inicializaciones

En esta fase se realiza la conexión con los hosts y la creación e inicialización de las estructuras de datos necesarias para la orquestación. También se alinean dichos datos con la Base de datos.

```
L1 = GetContainers(Node1)
L2 = GetContainers(Node2)
BD = GetContBDData()
OrqData = CompareStatus(L1,L2,BD)
UpdateContBD(OrqData)
```

Se almacena en una lista los contenedores que están actualmente activos de cada nodo. Se sacan los datos que tenía la Base de datos sobre el estado de dichos contenedores y se compara. Esto devuelve una lista ya actualizada sobre el estado actual de los contenedores, la cual es la que dispondrá el orquestador para realizar sus funciones. A partir de dicha lista, se actualiza la BD en consecuencia.

8.4.2. 1a Fase: Comprobación de los nodos

A continuación, se comprueba el estado de los nodos, es decir, si están levantados o no, y en caso afirmativo, se obtienen estadísticas como su uso de CPU y Memoria RAM en %.

```
if (ping(Node))
    data[node] = getCPU(Node)
    data[node] = getMEM(Node)
```

Si los nodos responden a ping, se lanzan las funciones para obtener sus datos en threads aparte, evitando así la ralentización del programa. A esta función se la llama una vez por nodo.

8.4.3. 2a Fase: Comprobación de los contenedores

Lo siguiente es realizar una actualización del estado de los contenedores, es decir, si por algún casual se han caído, o en la anterior iteración del orquestador se levantaron y ahora debe actualizarse el estado.

```
L = GetContainers(Node1)
BD = GetContBDData()
OrqData = CompareData(L, BD, OrqData)
UpdateContBD(OrqData)
for (each Cont in OrqData)
    Stats[Cont] = getStats(Cont)
    Details[Cont] = getDetails(Cont)
```

Primeramente, se obtiene la lista de contenedores que actualmente están activos según la API de Docker. Esta lista se compara con los datos que tiene la Base de Datos y los que tiene el orquestador en sus estructuras en ese instante. Esto actualiza la estructura del orquestador. Con esta estructura, se actualizan los datos que hay en la Base de Datos. Una vez hecho eso, y también mediante threads distintos, se obtienen las estadísticas y los detalles de cada contenedor. A esta función también se la llama una vez por nodo.

8.4.4. 3a Fase: Actualización de los datos

En esta fase se espera a que terminen todos los threads lanzados anteriormente y se actualizan los datos en la Base de Datos. En esta fase también se realiza la migración de los contenedores en caso de caída.

```
while (numThreads() != 1)
if (not NodeAlive(Node1))
        if (needMigration(Node1))
                migrateOnCritical(Node1)
if (not NodeAlive(Node2))
        if (needMigration(Node2))
                migrateOnCritical(Node2)
updateNodeBD(Node1)
updateNodeBD(Node2)
for (each Cont in orqCont)
        updateContBD(Stats[Cont], Details[Cont])
```

Los threads lanzados en las anteriores fases disponen de un tiempo de timeout. Si dicho tiempo expira, los valores que pueden contener las estructuras de datos pueden no ser correctos. El principal motivo por el que los threads finalicen debido a time out es a causa de una caída o un fallo de comunicación en el nodo. Por tanto, una vez todos los threads han finalizado, se realiza una comprobación de dicha situación y, en el caso de que efectivamente algún nodo no se encuentre disponible, se migren los contenedores que se ejecutaban en él. A continuación, se actualizan las Bases de Datos con los nuevos datos sobre los nodos y contenedores.

8.4.5. 4a Fase: Comprobación de las órdenes

El usuario también tiene la posibilidad de ejecutar ciertas órdenes desde la interfaz web del orquestador. En esta fase, se comprueba si existe alguna orden pendiente. En caso afirmativo, se realiza la acción correspondiente.

```
BD = OrdBDData()
for (each order, Cont in BD)
        if (order = Start)
                result = Start(Cont)
        if (order = Stop)
                result = Stop(Cont)
        if (order = Migrate)
                result = MigrateOnDemand(Cont)
        updateOrdBD(result, Cont)
```

Se recoge la orden que exista en la Base de Datos, si es que la hay, y se procesa. La realización de las órdenes también tiene un tiempo de timeout, por tanto, al finalizar, se actualiza la Base de Datos con el resultado de la ejecución de la orden. A esta función también se la llama una vez por nodo.

8.4.6. 5a Fase: Comprobación de la saturación

Por último, es el momento de comprobar si alguno o algunos de los contenedores están saturando algún nodo. En tal caso, se realiza una migración de los mismos al otro nodo, para tratar de compensar la carga en ambos.

```
if (NodeStats[Node].CPU > x)
        contC = max(Stats.CPU)
        migrateOnSaturation(contC)
if (NodeStats[Node].MEM > x)
        contM = max(Stats.MEM)
        migrateOnSaturation(contM)
```

Se comprueba tanto uso de CPU como de Memoria por parte del nodo. En caso que alguno supere un límite establecido, se busca que contenedor es el que más está saturando el recurso en cuestión. Una vez encontrado, se migra el contenedor al otro nodo. A esta función también se la llama una vez por nodo.

8.4.7. 6a Fase: Situación crítica

La mayoría de las fases anteriores solo se realizaba si el nodo estaba activo. Pero, en caso de que ninguno de los dos nodos estuviese activo, al final de la iteración del orquestador, se pasa a esta fase auxiliar en la que se gestiona dicha situación.

```
while (not ping(Node1) or not ping(Node2))
if ping(Node1)
        raiseAllCont(Node1)
else if ping(Node2)
        raiseAllCont(Node2)
```

En esta fase, se espera a que uno de los dos nodos conteste. El nodo que levante primero será el host para todos los contenedores que estaban activos antes de esta situación.

8.5. IMPLEMENTACIÓN DE LA INTERFAZ WEB

El desarrollo de la aplicación web combina 4 lenguajes de programación de páginas web: HTML, CSS, Javascript y PHP.

La estructura básica, es decir, el esqueleto de la web, se realiza mediante las funciones típicas de HTML. A esto se le añaden las facilidades de Bootstrap 4 de CSS para que la web tenga un diseño más estético y amigable. Mediante PHP se crean las estructuras de tamaño dinámico, como son las tablas con el listado de contenedores. También se realizan las consultas a la base de datos para obtener y mostrar la información tanto de los contenedores como de los nodos.

Por último, con Javascript se implementan los elementos más complejos de la web, como son los gráficos (gracias a la librería Graph.js) o los acelerómetros. También mediante Javascript se realizan las funciones que llaman a los scripts de php de actualización de datos, que, a su vez, estas funciones son llamadas por otra función en Javascript que está programada para ejecutarse cada dos segundos, ya que, como previamente se ha comentado, es el tiempo en que tardan en actualizarse los datos. A continuación, se muestran las capturas de pantalla de la interfaz web diseñada.

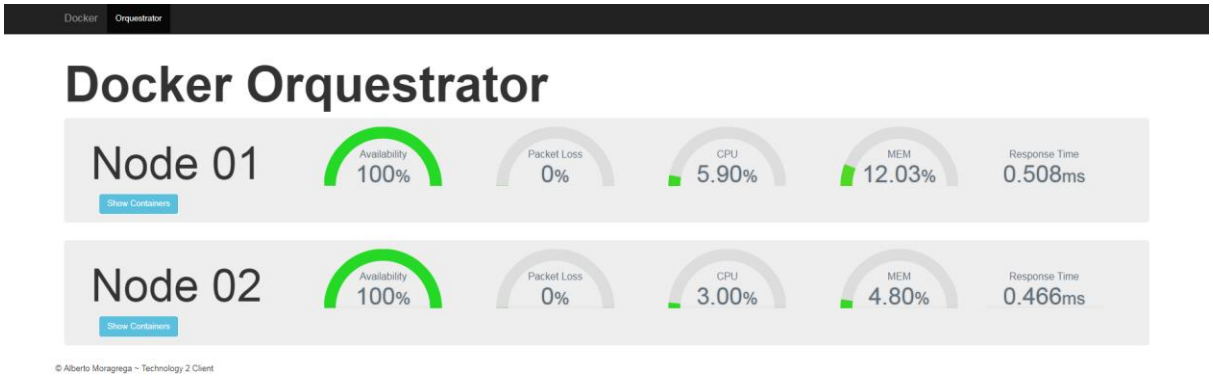


Figura 16: Pantalla inicial de la interfaz del orquestador.

En esta captura observamos la web en cuanto se accede. El listado de contenedores por cada nodo se encuentra oculto para evitar la saturación y facilitar la lectura de la página. También se muestran, representados como acelerómetros, las estadísticas de los nodos (disponibilidad, paquetes perdidos, %CPU, %MEM y tiempo de respuesta).

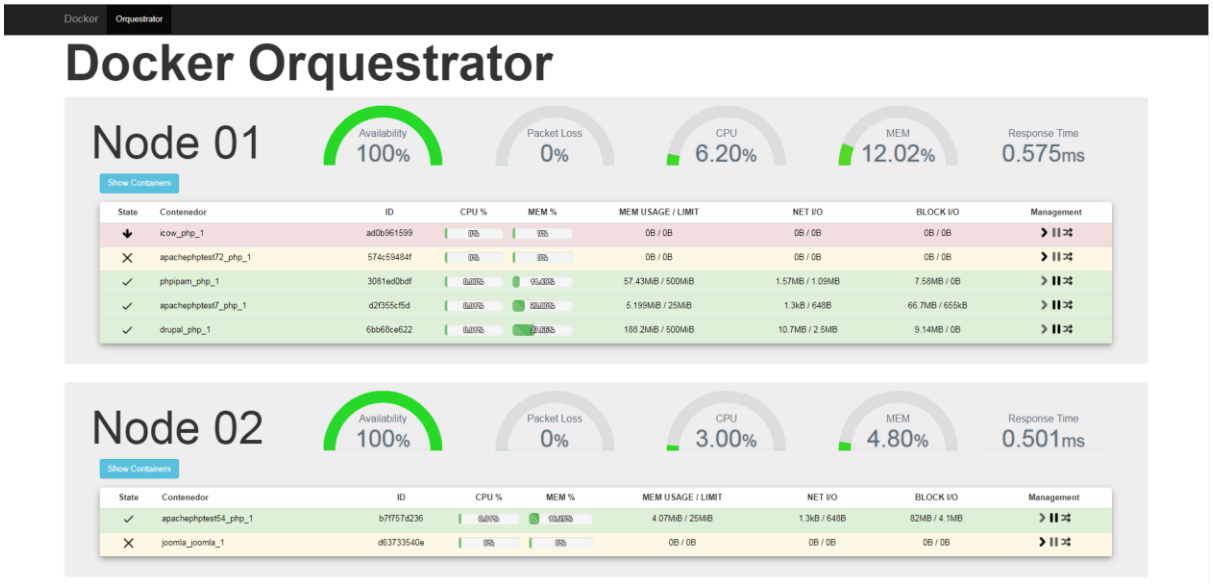


Figura 17: Detalle de los contenedores por nodo.

Aquí se muestra un posible escenario del orquestador. Cada fila del listado corresponde a un contenedor. Por cada uno se detalla su estado (verde si está corriendo, amarillo si está detenido y rojo si se encuentra caído), su nombre, ID, %CPU, %MEM, el uso de la misma respecto del total, el tráfico de red y el tráfico

con el disco. Por último, cada uno tiene tres botones con los cuales se arranca, detiene y migra el contenedor bajo demanda.

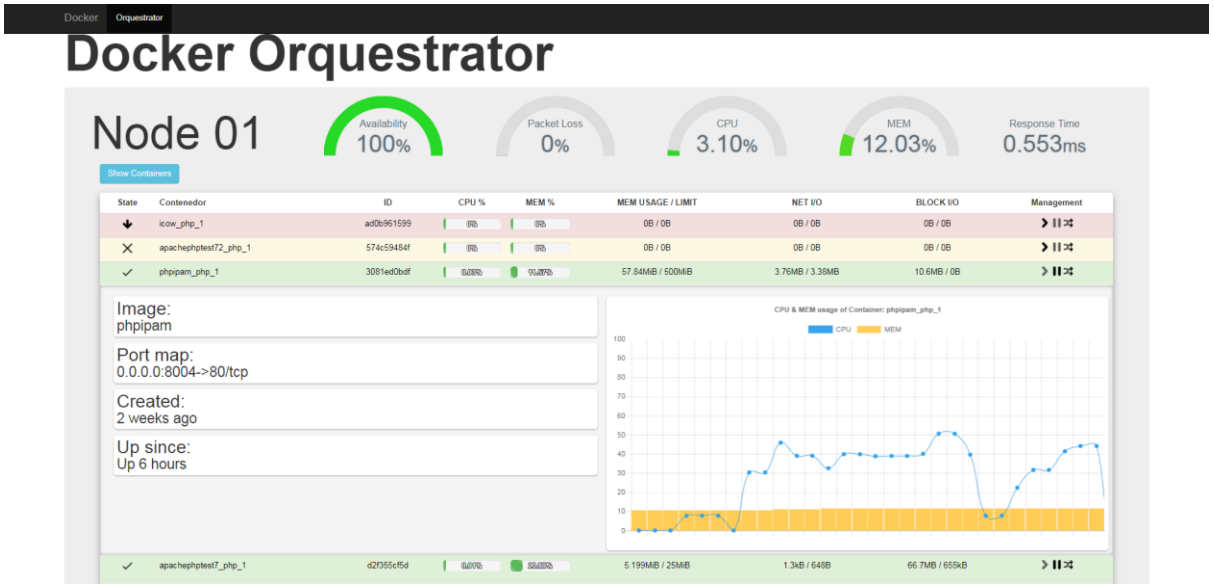


Figura 18: Detalle de un contenedor en un nodo.

Haciendo click en uno de los contenedores, se muestran sus detalles (Imagen que está ejecutando, puerto en el cual se encuentra, fecha de creación y tiempo levantado), así como un gráfico a tiempo real del uso de CPU y MEM del contenedor.

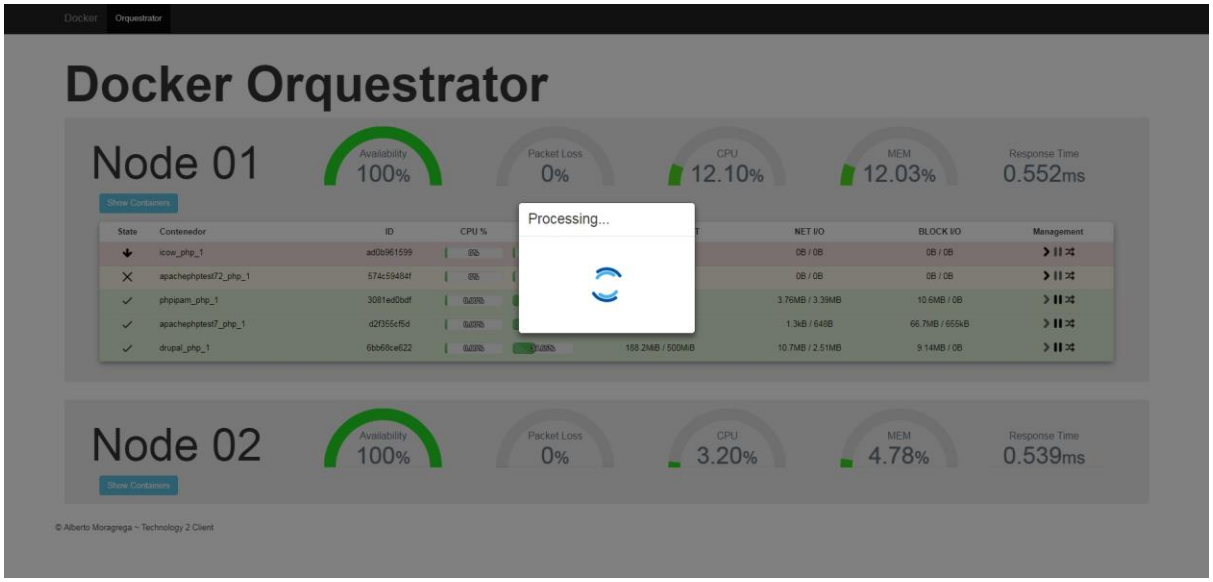


Figura 19: Pantalla de espera durante la realización de una acción.

Al ejecutar una orden, inmediatamente aparece una pantalla de espera, indicándonos sobre el estado de la petición que se acaba de efectuar. Esto hace que sólo se pueda ejecutar una orden cada vez. En caso de que la orden no

pudiese efectuarse, la pantalla nos lo indicaría, dejando todo como estaba antes de la última orden.

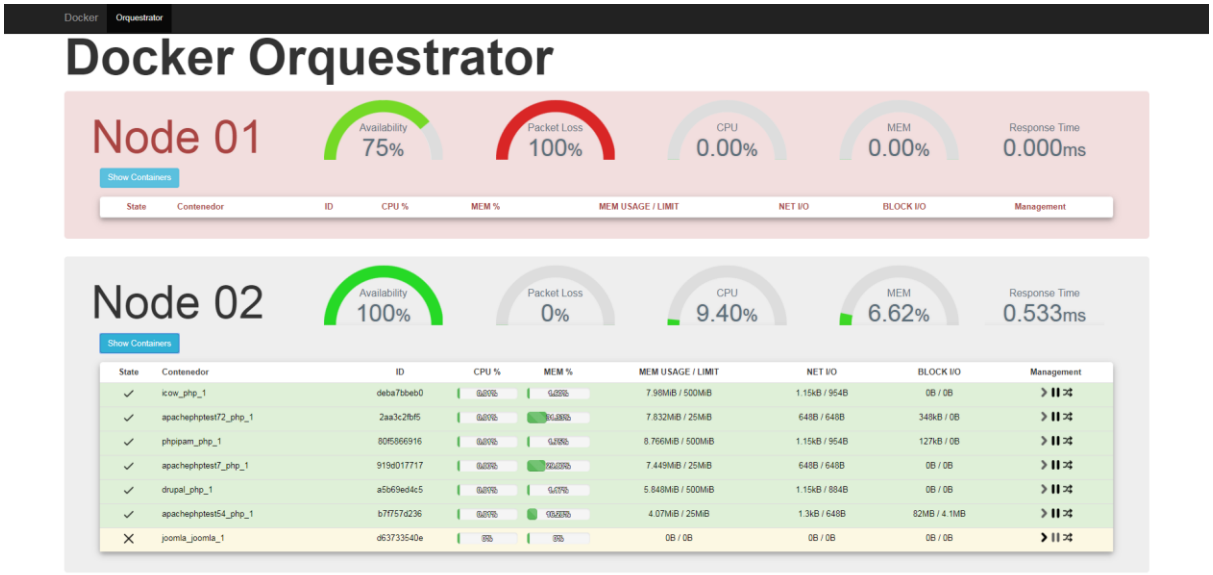


Figura 20: Detección de un nodo caído.

Por último, en caso de que algún nodo esté caído, éste aparecerá marcado en rojo, y automáticamente los contenedores que se encontraban en él migrarán al siguiente gracias a la autonomía diseñada por parte del programa del orquestador.

La comunicación entre la aplicación web y el software del orquestador se realiza mediante una base de datos mySQL. Las tablas utilizadas para dicha comunicación se detallan en capítulos posteriores.

El programa del orquestador está programado en lenguaje Python, ya que, la API oficial de Docker solo se encuentra en dicho lenguaje o en Go. Este programa se encarga de recopilar los datos para mostrarlos vía web y de ejecutar las órdenes que le envía la misma. Como se ha podido observar en las imágenes anteriores, también dispone de autonomía para realizar migraciones de contenedores cuando el nodo está caído o cuando existe mucha carga en uno de ellos.

8.6. COMUNICACIÓN ORQUESTADOR-WEB

Tal y como se ha nombrado en los dos apartados anteriores, la comunicación entre la web y el orquestador se realiza mediante una Base de Datos mySQL. Dicha base de datos contiene 3 tablas, que se detallan a continuación:

- NodeStatus [Node, Availability, PacketLoss, ResponseTime, NodeUp, CPU, MEM]: Esta tabla es la que se encarga de almacenar la información sobre el estado del nodo.
- ContainerStatus [Node, ID, ContName, State, CPUperc, MEM, MEMperc, NetIO, BlockIO, Image, Port, Creation, Uptime]: En esta tabla se encuentra el estado, estadísticas y detalles de cada contenedor.
- OrquestratorOrders [Node, ContName, Orders]: Ésta es la tabla encargada de enviar las acciones que se realizan vía web para que el programa del orquestador las ejecute y devuelva el estado de dicha acción.

Estas tablas son accesibles tanto para la web como para el orquestador y es el camino por el cual comparten los datos. Es decir, cuando el usuario envía una orden, se actualiza dicha orden en la base de datos y el orquestador, en la fase de comprobación de las órdenes, consulta esa tabla y realiza la acción pertinente.

Por el lado contrario, el orquestador está constantemente actualizando el estado de los contenedores y el nodo en sus respectivas tablas, las cuales son leídas por la web para mostrarlas por pantalla.

9. Resultados experimentales

Para comprobar la viabilidad de la infraestructura propuesta se toman datos para realizar una representación gráfica de los mismos, con tal que la comparación sea lo más visual y entendible posible.

9.1 DEFINICIÓN DE LAS MÉTRICAS

Las métricas que se han definido para realizar la comparación y así obtener los resultados son las siguientes:

- Uso de CPU, en %.
- Uso de Disco, en KBps.
- Consumo de Memoria RAM, en KB.
- Tráfico de red, en KBps.
- Consumo eléctrico, en Watts.
- Disponibilidad, en Segundos.
- Hardware, en unidades.

9.2. REPRESENTACIÓN GRÁFICA DE LOS DATOS

Las pruebas se realizan sometiendo cada uno de los tres modelos (Tradicional, Docker con orquestador y Docker sin orquestador) a una serie de accesos y uso de las aplicaciones web que contienen. Ambos modelos se encuentran en la misma máquina física y ejecutan las mismas webs. En la representación se han unificado las métricas de cada máquina virtual de un mismo modelo, con tal de poder realizar una comparación global a nivel de infraestructura.

La recolecta de datos para las representaciones gráficas se ha tomado acuerdo al siguiente escenario:

- La infraestructura tradicional se compone de dos servidores virtuales que alojan una y dos aplicaciones web respectivamente.
- En cuanto a la infraestructura Docker, se han migrado todos los contenedores al nodo número uno, sin embargo, el resto de contenedores de pruebas se han dejado activos en el segundo nodo, aunque no se realicen accesos, con tal de poder valorar mejor el incremento de carga que supone el uso del

orquestador, ya que, con él desactivado, la aportación de ese nodo a los datos es casi irrelevante.

- Para tener una visión más clara del beneficio que supone Docker, se han recolectado datos con el orquestador desarrollado activo y detenido.

A continuación, se muestran los datos de los resultados experimentales realizados en forma de gráficos para facilitar su comprensión.

- CPU:

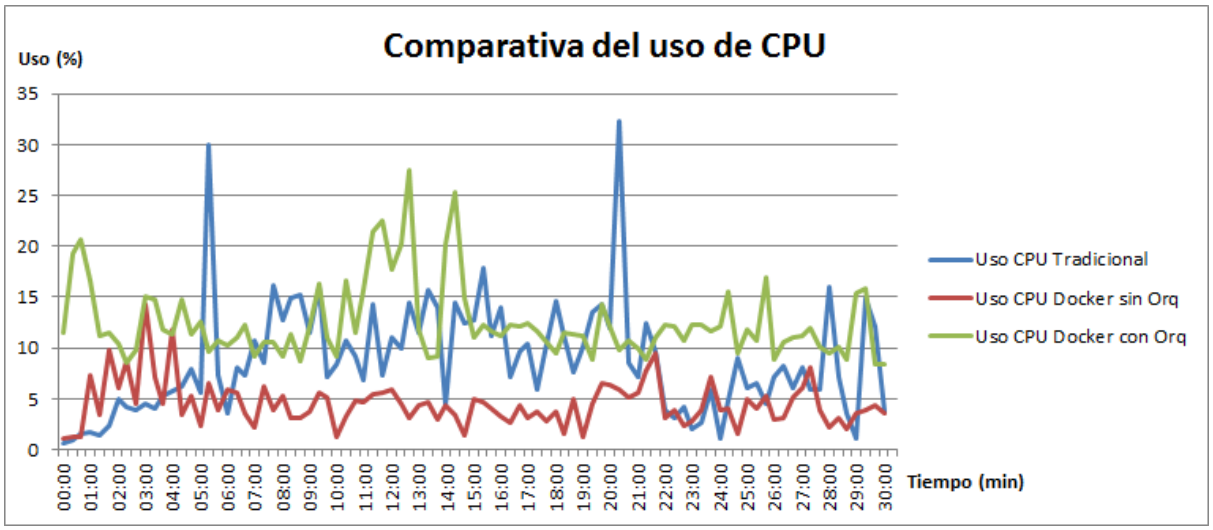


Figura 21: Gráfico comparativo del uso de CPU.

Una de las primeras ideas que se pueden extraer del gráfico es que existe un cierto ahorro en cuanto a uso de CPU en una infraestructura virtualizada y una virtualizada con containerización pero que, sin embargo, este uso se ve aumentado debido al orquestador implementado. A pesar de esto, con alguna optimización, el añadido al consumo de CPU que corresponde al orquestador podría no ser un problema a gran escala, ya que el consumo de un elevado número de máquinas virtuales seguirá siendo mayor. Otro punto a destacar sobre el orquestador es que está consultando el estado de los nodos cada 2 segundos, lo cual podría llegar a crear saturación.

Este ahorro en CPU es debido a que la implementación de Docker permite repartir el uso de la CPU entre los contenedores, ya que todos comparten el mismo kernel de Linux para su uso, lo cual hace que no haya gasto en CPU extraido del uso del Sistema Operativo.

- Disco:

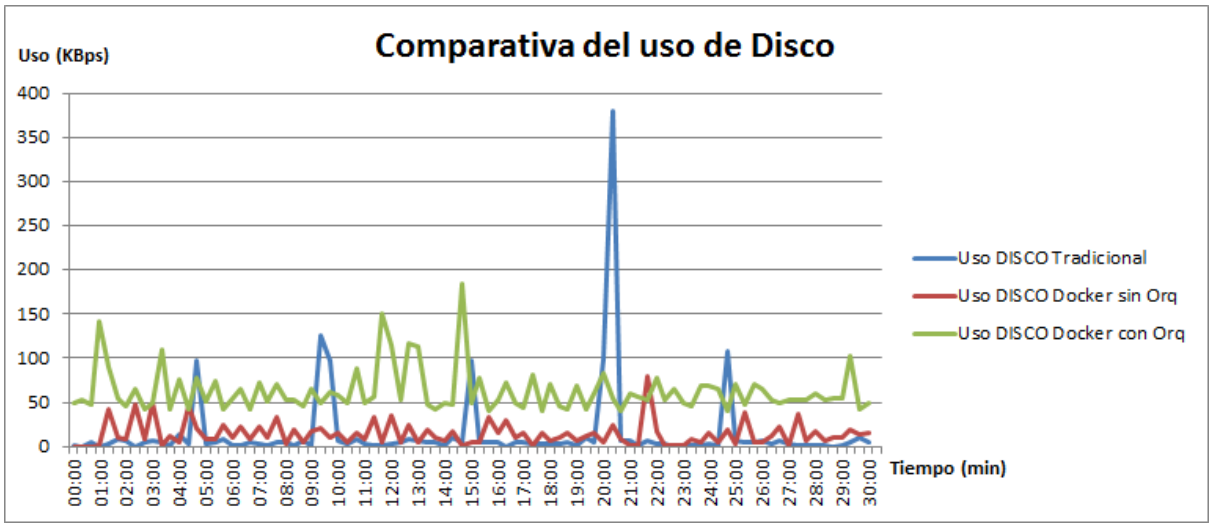


Figura 22: Gráfico comparativo del uso de disco.

Observando el gráfico se puede extraer como Docker tiende a tener un mayor consumo de disco, aunque esta diferencia sea mínima. Esto es debido que Docker crea dos particiones lógicas en disco por cada contenedor que tiene activo, lo cual hace que aumente el tráfico en el mismo. Esto se puede observar mediante el comando *df -h:*

```
[root@dockerprod01 ~]# df -h
S.ficheros                                   Tamaño Usados  Disp Uso% Montado en
/dev/sda1                                      16G   4,6G   11G  31% /
devtmpfs                                      2,0G      0  2,0G   0% /dev
tmpfs                                         2,0G      0  2,0G   0% /dev/shm
tmpfs                                         2,0G    57M  1,9G   3% /run
tmpfs                                         2,0G      0  2,0G   0% /sys/fs/cgroup
/dev/sdc1                                      20G   148M   19G   1% /opt
t2cnashq01.t2client.lan:/nfs/docker_content   197G   119G   78G  61% /opt/docker
overlay                                        16G   4,6G   11G  31% /var/lib/docker/overlay2/c8d5bfade052cf116
shm                                            64M      0   64M   0% /var/lib/docker/containers/0a2a2291c3b5827
overlay                                        16G   4,6G   11G  31% /var/lib/docker/overlay2/46a7c1eed8c0f1e8b
shm                                            64M      0   64M   0% /var/lib/docker/containers/3081ed0bdf54296
overlay                                        16G   4,6G   11G  31% /var/lib/docker/overlay2/7452a27b447c4fb1a
shm                                            64M      0   64M   0% /var/lib/docker/containers/6bb68ce6224a462
tmpfs                                         396M      0  396M   0% /run/user/0
```

Figura 23: Ejemplo de particiones en uno de los nodos.

Al contrario que sucedía en la métrica de la CPU, con el orquestador activado el consumo de disco es probable que aumente un poco más cuantos más contenedores se encuentren en un nodo, debido a que se crean más particiones lógicas que derivan en un aumento de accesos al disco físico.

- Memoria:

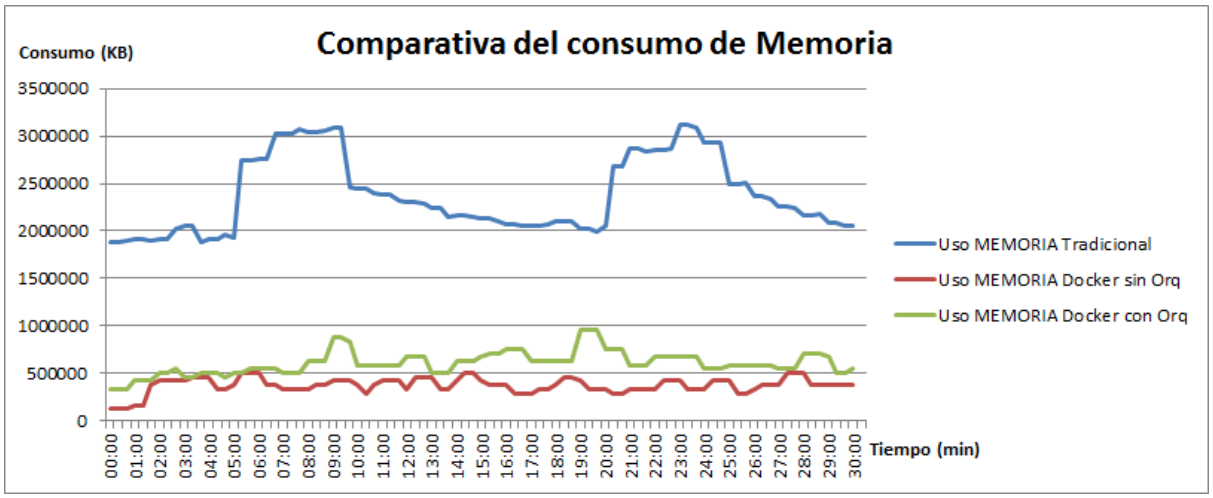


Figura 24: Gráfico comparativo del consumo de memoria.

Al contrario de lo que sucedía con disco, Docker si demuestra tener un menor consumo de memoria RAM. Esto es debido a que Docker no almacena los cambios que se realizan en las imágenes mientras están en ejecución. Es decir, simplemente realiza una operación de diferencia con lo que existe en el contenedor y la imagen. Esta diferencia es lo que realmente se guarda en memoria, lo cual hace que la cantidad de datos a almacenar se reduzca drásticamente.

- Tráfico de red:

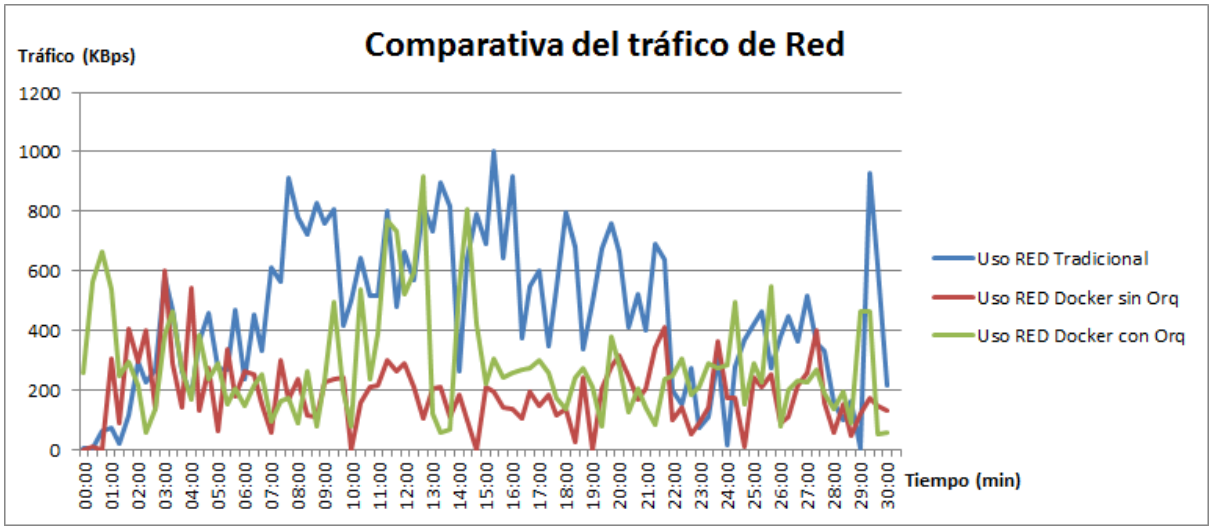


Figura 25: Gráfico comparativo del tráfico de red.

En este caso tampoco existe una diferencia notable entre tener el orquestador activado o desactivado, ya que las acciones que envía a los nodos son bastante simples.

Sin embargo, si existe una diferencia entre la infraestructura clásica y Docker, debido a que los contenedores sólo contienen lo estrictamente necesario para su uso, con lo cual, se evita que otros programas o elementos del Sistema Operativo utilicen la red.

- Energía:

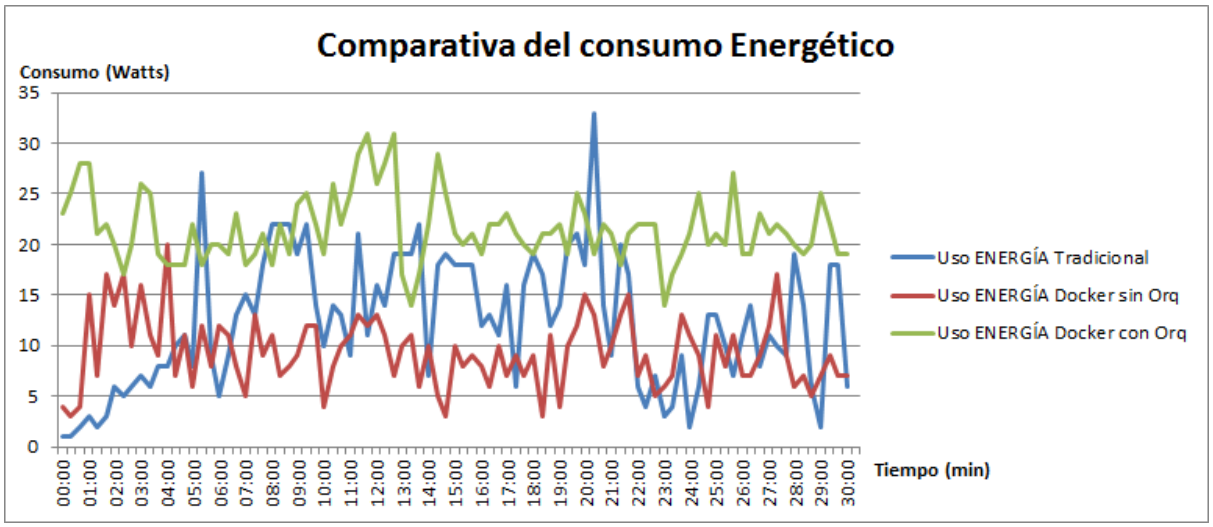


Figura 26: Gráfico comparativo del consumo energético.

Una situación similar a la métrica de la CPU ocurre en cuanto al consumo energético de las infraestructuras. El modelo con Docker demuestra ser algo más eficiente que el modelo tradicional, sin embargo, el uso del orquestador aumenta considerablemente la energía consumida. A pesar de esto, al igual que con la CPU, al ser siempre las mismas las operaciones que realiza el orquestador, a gran escala si se vería una clara reducción en energía, debido a la reducción de nodos que Docker permite.

- Disponibilidad:

Mientras que el tiempo de despliegue de las aplicaciones en una máquina virtual tradicional se ha comprobado que varía de entre 15 a 17 segundos, levantar un contenedor Docker con su aplicación tarda tan solo entre 2 y 3 segundos. Esto, sumado al segundo nodo que propone este modelo de infraestructura, asegura que el tiempo de indisponibilidad de los servicios web es el mínimo, ya que el tiempo de migración de los contenedores a otro host es tan solo de 3 a 6 segundos. La herramienta HA Proxy ya se encargaría de la redirección del tráfico al nuevo nodo donde se aloja el contenedor.

Por contra, en el modelo de infraestructura tradicional, el tiempo de indisponibilidad sería mayor, variando según si el problema que ha causado la caída del nodo es de mayor o menor complejidad.

- Hardware:

En el caso de esta pequeña aproximación, el número de nodos dedicados a aplicaciones web en ambas infraestructuras era el mismo, sin embargo, esto puede no ser así y requerir nodos independientes para cada aplicación web, problema que se ve solventado con la independencia y el aislamiento que ofrece Docker. Es decir, a gran escala, el número de máquinas necesarias con esta implementación podría verse reducido hasta en un 95%. [40]

9.3. RESULTADOS OBTENIDOS

- El uso de CPU se ve reducido en un 48,68% de media en la versión sin orquestador respecto de la tradicional. Por contra, el uso de CPU con el orquestador implementado aumenta en un 42,22% respecto de la tradicional.
- En cuanto al uso de disco, la infraestructura propuesta contra la tradicional tan solo reporta un 2,98% de beneficio. No obstante, con el orquestador activado, el uso de disco aumenta considerablemente respecto de la original.
- En lo que respecta al uso de memoria, ambas variantes de la nueva infraestructura reportan beneficios de un 84,24% con el orquestador inactivo y un 74,51% con el orquestador activo respecto de la tradicional.
- Con el uso de la red existe una mejora respecto de la original de 59,13% en el caso en que el orquestador se encuentre inactivo, y una mejora de un 38,81% de media en caso de que el orquestador se encuentre activo.
- En cuanto a energía, con el nuevo modelo sin uso del orquestador se consigue un ahorro de un 24,19% respecto del modelo tradicional. Sin embargo, con el orquestador activado, el consumo aumenta en un 70.64% respecto del modelo original.
- El tiempo de despliegue respecto de la versión tradicional de la infraestructura ha mejorado en un 84,37% con este nuevo modelo propuesto.
- En este estudio no se ha realizado ningún incremento o decremento en el número de máquinas necesarias para implementar la nueva infraestructura.

10. Conclusiones

Mediante la implementación realizada y el estudio detallado en el punto anterior, se puede concluir que, en mayor o menor medida, se han cumplido todos los objetivos del proyecto.

- Se ha conseguido containerizar el pequeño grupo inicial de aplicaciones que se había propuesto desde el inicio del proyecto. Esta tarea ha resultado ser relativamente sencilla, lo cual indica que Docker es una herramienta fácil y agradable de aprender.
- El modelo de infraestructura implementado ha sido puesto en producción sin problemas. Todos los nodos cumplen su función de forma esperada.
- Todas las funcionalidades del orquestador han sido implementadas exitosamente. El orquestador cumple las funcionalidades para las que ha sido diseñado.
- La aplicación web funciona como se espera. Su interfaz es intuitiva y muy visual, facilitando al usuario la monitorización y gestión de los contenedores en cada nodo.
- El rendimiento de Docker respecto a la infraestructura tradicional ha sido el esperado. El orquestador añade un consumo notable a los recursos, pero a gran escala probablemente no sea considerable. Independientemente de esto, este aspecto del orquestador abre la puerta a futuras mejoras y novedades.

Realizando una comparación con la infraestructura anterior que se disponía, el nuevo modelo añade diversas mejoras a tener en cuenta:

- Portabilidad: Las aplicaciones se montan en los contenedores una sola vez y, si su funcionamiento es el correcto, seguirán funcionando del mismo modo independientemente del entorno en el que se encuentren. Esto elimina por completo los problemas de compatibilidad derivados de un cambio de sistema operativo en las máquinas virtuales, o en actualizaciones del mismo, etc.
- Fiabilidad: Los contenedores, al ser entornos aislados, compactos y compatibles, son fácilmente migrables. Esto hace que, ante una posible caída de un nodo, el contenedor pueda levantarse de forma rápida y eficaz en

cualquier otro nodo, reduciendo así el tiempo de downtime para el usuario de la aplicación que se esté ejecutando.

- Unificación: Los contenedores son mucho más ligeros que una máquina virtual completa, por tanto, en un mismo nodo se pueden llegar a ejecutar un número considerable de aplicaciones, cada una en su propio contenedor. Esto facilita la tarea de la administración de sistemas y abre la posibilidad de poder utilizar otras máquinas que anteriormente estaban dedicadas a estas aplicaciones a nuevas funcionalidades o incluso a no hacer uso de ellas para reducir en consumo y alargar la vida útil de los componentes.

Basándose en los aspectos analizados en el estudio estadístico, se puede observar una mejoría en la mayoría de aspectos del nuevo modelo propuesto. Concretamente, el modelo con el orquestador activado reporta beneficios en todos los recursos, lo cual reafirma las hipótesis sobre el uso de recursos de los contenedores y Docker.

Por contra, como se ha podido observar en el estudio estadístico, el uso del orquestador añade un cierto consumo en algunos de recursos de la máquina. Éste es el siguiente punto a tratar en futuras mejoras de la implementación del mismo. Sin embargo, un mayor número de servidores permitiría escalar correctamente y el coste computacional introducido por el orquestador sería cada vez menos relevante. Por lo tanto, pese a que aún se puede optimizar el orquestador, recomendamos este tipo de instalación para un número considerable de servicios o máquinas virtuales

Otra mejora destacable en cuanto a implementación del orquestador es la propia encapsulación del mismo en un contenedor. Con esto, se conseguiría que, si por algún motivo el nodo que fallase fuese el de orquestación, el propio orquestador migrase a cualquiera de los otros nodos, gestionando los contenedores desde ahí mientras el nodo se encuentre indisponible.

Otras propuestas para futuras mejoras en cuanto a la infraestructura incluyen la posibilidad de mantener siempre todos los contenedores de una misma imagen activos, de manera que el proxy fuese el encargado de redirigir el tráfico para realizar un balanceo de carga siempre y cuando todos los nodos estuviesen disponibles.

Por último, destacar que tal y como se encuentra implementada la infraestructura, ésta está completamente lista para ser escalable. Gracias a que los ficheros de configuración y los datos que necesitan las aplicaciones están compartidos en una

máquina externa, añadir un nuevo nodo host de Docker a la infraestructura requiere de muy poca configuración adicional.

Concluyendo, la containerización de aplicaciones es una tecnología que tendrá un gran auge en el futuro. Cada vez son más las empresas, grandes y pequeñas, que se benefician de sus cualidades. La virtualización de los sistemas supuso un cambio en la forma de utilizar las máquinas, hasta el punto que actualmente es de las tecnologías más usadas. Probablemente, en el futuro la containerización sea tan habitual como hoy lo es la virtualización de los sistemas.

Referencias

1. Wikipedia. *Ley de Moore.* [En línea] 1 de Diciembre de 2017. https://es.wikipedia.org/wiki/Ley_de_Moore.

2. **Jones, M.** IMB. *Virtualización de aplicaciones, pasado y futuro.* [En línea] IBM, 29 de Julio de 2011. https://www.ibm.com/developerworks/ssa/linux/library/l-virtual-machine-architectures/index.html.

3. Docker. *Docker Commercial Customers.* [En línea] 2018. https://www.docker.com/customers.

4. Aqua. *A Brief History of Containers: From the 1970s to 2017.* [En línea] Aquasec, 21 de Marzo de 2018. https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016.

5. Wikipedia. *UNIX Versión 7.* [En línea] 20 de Diciembre de 2017. https://es.wikipedia.org/wiki/UNIX_Versi%C3%B3n_7.

6. FreeBSD. [En línea] 2018. https://www.freebsd.org/.

7. Linux VServer. *Overview.* [En línea] 22 de Abril de 2013. http://linux-vserver.org/Overview.

8. Oracle. *Solaris Containers.* [En línea] http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html.

9. Wikipedia. *OpenVZ.* [En línea] 21 de Noviembre de 2017. https://es.wikipedia.org/wiki/OpenVZ.

10. Wikipedia. *cgroups.* [En línea] 23 de Junio de 2018. https://en.wikipedia.org/wiki/Cgroups.

11. Linux Containers. *LXC.* [En línea] https://linuxcontainers.org/.

12. Wikipedia. *Cloud Foundry.* [En línea] 16 de Enero de 2018. https://es.wikipedia.org/wiki/Cloud_Foundry.

13. Wikipedia. *lmctfy.* [En línea] 10 de Abril de 2018. https://en.wikipedia.org/wiki/Lmctfy.

14. Docker. [En línea] 2018. https://www.docker.com/.

15. Wikipedia. *Docker.* [En línea] 21 de Junio de 2018. https://es.wikipedia.org/wiki/Docker.

16. Kubernetes. [En línea] 2018. https://kubernetes.io/.

17. Rancher. [En línea] https://rancher.com/.

18. VMware. [En línea] 2018. https://www.vmware.com/.

19. CentOS. *The CentOS Project.* [En línea] 2018. https://centos.org/.

20. Putty. [En línea] https://www.putty.org/.

21. Google. *Google Drive.* [En línea] https://www.google.com/intl/es_ALL/drive/.

22. WinSCP. [En línea] https://winscp.net/eng/docs/lang:es.

23. Sonic Wall. [En línea] 2018. https://www.sonicwall.com/en-us/home.

24. Python. *Python 2.7.0 Release.* [En línea] 2001-2008. https://www.python.org/download/releases/2.7/.

25. MySQL. [En línea] 2018. https://www.mysql.com/.

26. Wikipedia. *HTML.* [En línea] 7 de Junio de 2018. https://es.wikipedia.org/wiki/HTML.

27. PHP. [En línea] 2001-2018. http://php.net/.

28. Wikipedia. *Hoja de estilos en cascada.* [En línea] 16 de Mayo de 2018. https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada.

29. Wikipedia. *Javascript.* [En línea] 5 de Junio de 2018. https://es.wikipedia.org/wiki/JavaScript.

30. Trello. [En línea] 2018. https://trello.com/.

31. VMware. *Vcenter.* [En línea] 2018. https://www.vmware.com/es/products/vcenter-server.html.

32. Wikipedia. *Contenedores de software.* [En línea] 5 de Junio de 2018. https://es.wikipedia.org/wiki/Contenedores_de_software.

33. Docker. *What is a container.* [En línea] 2018. https://www.docker.com/what-container.

34. Linux. *Docker: A 'Shipping Container' for Linux Code.* [En línea] 1 de Agosto de 2013. https://www.linux.com/news/docker-shipping-container-linux-code.

35. Docker documentation. *Dockerfile reference.* [En línea] 2018. https://docs.docker.com/engine/reference/builder/.

36. Docker documentation. *Docker Compose.* [En línea] 2018. https://docs.docker.com/compose/.

37. Docker-py. *Docker SDK for Python.* [En línea] 2018. http://docker-py.readthedocs.io/en/stable/.

38. Docs Google. *Cuestionario de Estudiantes de Ingeniería Informática.* [En línea] https://docs.google.com/forms/d/e/1FAIpQLSelZixKIUFbCn1oVkd2JM3yxCc208E85RgKZclKd8eUu3GvBg/viewform.

39. EL INFORME DE SOSTENIBILIDAD DEL TFG. [aut. libro] Profesorado de GEP. UPC. Página 3. Figura 2.

40. Docker. *Infraestructure Optimization.* [En línea] 2018. https://www.docker.com/use-cases/infrastructure-optimization.

Anexo

ANEXO 1: Comandos de Docker utilizados

Ejecutar un contenedor con una imagen interactivamente

docker run -it “imagen”

Ejecutar un contenedor con una imagen en background

docker run -itd “imagen”

Parar un contenedor

docker stop “contenedor”

Arrancar un contenedor

docker start “contenedor” [Uno] / docker start $(docker ps -a -q) [todos]

Descargar una imagen

docker pull “imagen”

Descargar una versión concreta de una imagen

docker pull “imagen”:”ver.”

Ver imágenes descargadas

docker images

Ver contenedores corriendo

docker ps [Corriendo] / docker ps -a [Todos]

Acceder a un contenedor corriendo

docker exec -it “contenedor ID” bash

Salir de un contenedor sin cerrarlo

ctrl + p + ctrl + q

Ver las estadísticas de un contenedor

docker stats “contenedor”

Eliminar cont

docker rm “contenedor”

Eliminar imagen

docker rmi “imagen”